1.0 Quality Compression Optimization

The following sections deal with the improvement of the compressed image quality. As stated in the report introduction, we believe this is the last significant major problem to be addressed in the still imagery compression system.

1.1 Introduction

The issue is the mechanism which can be used to improve the quality of the image based upon the nature of the parameters used in the compression process. We think this is an important issue, and in our experimentation, we have found fairly striking results. For example, if we hold the compression ratio constant, we can compress an image and obtain on a subjective scale of C, where

- A is no observable defect,
- B is observable but not noticeable,
- C is quite noticeable, but not distracting from the image,
- D is very noticeable and detracts from the image,
- E is unacceptable.

If we now change the parameters by a hand optimization, we find we can change the subjective evaluation from a C to a B with the compression ratio left alone. We believe this is significant, in that it guarantees a compression system which is tailored for each image independently, rather than have each image compressed by the same set of parameters irrespective of the image content.

To review the process of compression, consider the following figure, Figure 49.

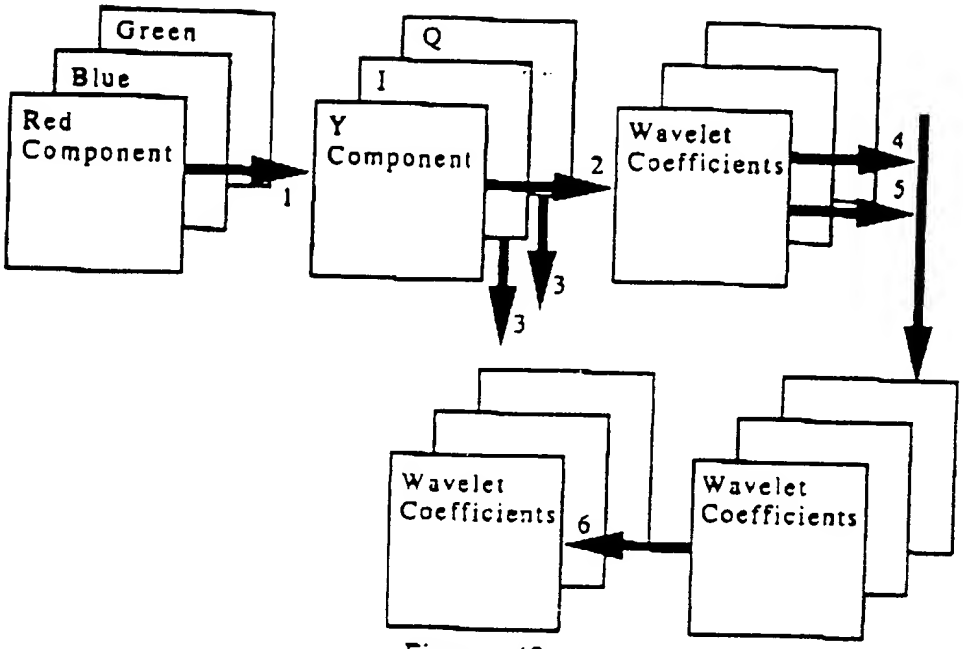


Figure 49
RGB Image in 24 Bit Color Depth Showing Transform to YIQ

Figure 49 shows the relationship between the RGB and YIQ color representation and the processes which take place in the algorithm as this data is transformed into the final lossy set of wavelet coefficients.. The processes which apply to the imagery are shown as heavy arrows with the process identifier shown as part of the arrow. If no process number is present, then the arrow is just a passive link between processes and data. The important point to note is that in color imagery, we deal with three images (one luminescence, and two color planes). The Y component is critical, while the IQ components are less sensitive to error introduced by the compression system.

The processes shown in Figure 49 are as follows:

- (1) Transform the RGB format into YIQ format, using long integers. This format is only an approximation of the YIQ format.

- (2) Transform the YIQ planes into a wavelet decomposition using our own integer wavelet transform. This produces the result shown in Figure 50, but for each plane.

- (3) We have shown an alternative process which is an optional down sampling for the IQ color planes. This down sampling may

be done once or twice to produce two image planes either one fourth or one sixteenth the size of the original plane. If this process is to be done, it will be accomplished prior to the wavelet transform of process (2).

- (4) Here the first step in the loss occurs. The wavelet coefficients are now quantized to a number of levels depending upon which quadrant is being processed, and the desired compression or quality factor.

- (5) Simultaneously with step (4), the wavelet coefficients are being matched against threshold values, and if the values are less than the established threshold values specified, then the resultant value is set to zero.

- (6) The last step in the process is to entropy compress the resultant coefficients using either Arithmetic, Run Length, or Huffman, of Huffman and Run Length combined. The key issue is the amount of compression desired against the invested time required to get that level of compression.

The issue now, irrespective of the down sampling or not of the IQ components, is the fact that we process the three planes into five levels in a decomposition as shown in Figure 50. From this figure, one can see a total of 16 regions (quadrants) which are defined by the numbers 1 through 16. Each of these sixteen quadrants have two additional parameters associated with them. The parameters define the quantization and threshold values for that particular quadrant. Since there are three planes for color (only one for gray level) the maximum number of parameters that the user can control is 96 -- 16 for quantization and 16 for thresholding for each of the three color layers. In the case of a gray level image, there are only parameters for one layer.

Our experience has shown the parameters for an image are very sensitive in some cases, and not in other cases. In order to measure this sensitivity, we generated the variance of the wavelet coefficients in the 16 quadrants. Table 1 provides these values for each of the three planes.

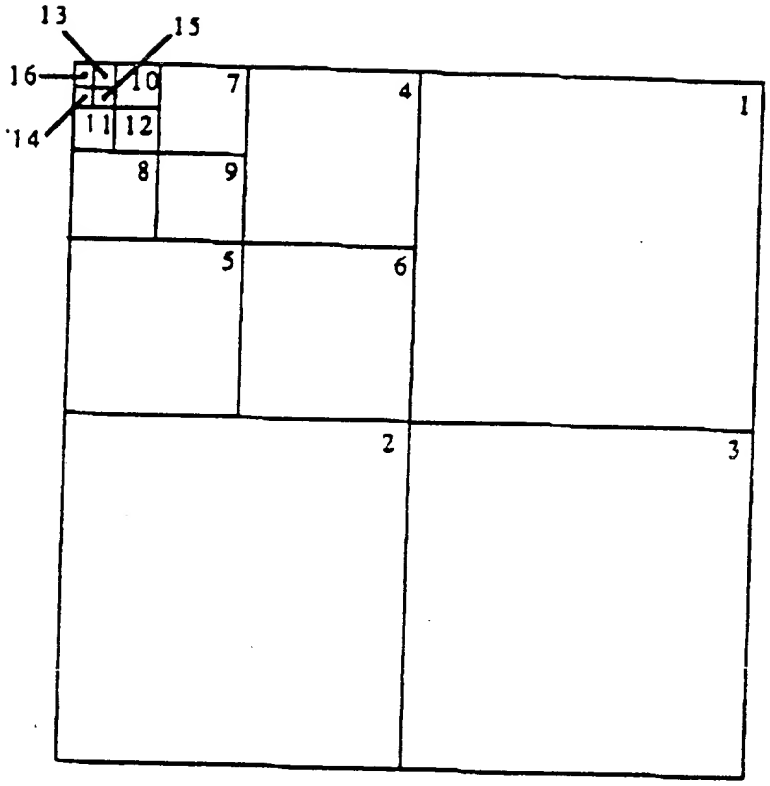


Figure 50
Decomposition of a Plane into Five Levels

Image	Lenna Wavelet Variance			Tulips Wavelet Variance		
Quadrant	Y	I	Q	Y	I	Q
1	13	4.7	1.4	164.2	27.3	6.6
2	6	3.2	1.1	134.6	24.9	5.0
3	1	1.3	0.6	28.9	4.6	1.1
4	56	5.8	1.7	276.1	57.7	12.9
5	23	4.5	1.1	274.3	67.3	12.4
6	10	2.8	0.8	124.8	22.7	5.0
7	128	11.4	2.6	225.7	73.2	13.7
8	55	7.4	1.1	298.1	107.3	16.2
9	37	3.3	1.0	114.2	35.5	6.8
10	253	27.5	4.1	174.8	73.9	10.7
11	75	11.8	1.5	285.1	128.2	14.9
12	64	6.5	1.1	107.9	47.8	6.7
13	499	41.7	10.1	135.4	75.6	8.7
14	138	11.2	2.5	245.0	144.0	11.2
15	156	12.2	2.0	89.3	49.1	5.1
16	1,4161	1,281.8	295.6	7,408.6	690.4	77.9

Table 1
Wavelet Coefficient Variance by Numbered Quadrant for Two Images

The clear information to be gained by the numbers in Table 1 is the images are quite different, and the quantization or threshold levels set for all images will only be an approximate solution at best. The optimum solution would be to have the quantization and threshold values set according to the variance values. Such settings could be found in a table with various ranges, and for each such range, the parameters of interest could be defined. This would give a more optimal solution, but still not the optimal solution. In order to get optimality, one would need to search over the variable space using the near optimal settings to find the actual best values for the parameters.

As the values in Table 1 grow, the implication is a finer mesh size for quantization. That is, the number of bits one wishes to allocate to the output could be varied by quadrant. Those quadrants with large variances will utilize more bits, while those with low variants will utilize fewer bits. In this way, the number of bits resulting from quantization will remain the same, but their allocation will differ depending upon the nature of the image.

1.2 Approach

The solution to the problem posed in the previous section is to determine how the ninety-six parameters interact with one another. The problem is a bit more sophisticated than just measuring parameters, however. The issue is with each parameter change, the compression ratio will change. If a compression ratio, or a quality factor which indirectly defines a compression ratio, is specified, then the user wants the compression ratio to remain identical over the changes in the parameters. In order to accomplish this, there are two parameters which we must monitor: PSNR (peak signal to noise ratio which is defined to be PSNR = 20 $\log_{10}$ (X/MSE) where the X is the average absolute value of the pixels in the after image and MSE is the mean squared error measured between the before and after image) and the compression ratio. The compression ratio must be held constant, and the PSNR needs to increase, and the way to increase the PSNR is to reduce the MSE.

The difficulty with this system as described is in many cases, small changes in the parameters introduce significant changes in the MSE. Also, we believe, the parameters are not independent. We have also seen images where the parameters can be changed in one way, then

altered, and the results are exactly the same. This indicates the optimal value is not a single point, but rather something like a plane with little slope.

1.3 Status

We have established the rules under which the optimal solution will need to exist, and are at the moment writing software to measure the variance within the Lightning Strike environment. Once this is done, we will being examining many images to see how close we can determine the optimal parameters for a defined set of variances.

CIS-2 Image Compression Algorithm

HONGYANG CHAO

Part I: Brief Review of LSIC 3.0

1. Introduction

CIS-2 (temporary name), which has been being used in **Lightning Strike 3.0** image compression software, is a wavelet based image compression algorithm. CIS-2 has following inventions:

- Integer reversible wavelet algorithm with Property of Precision Preservation;
- Subband oriented quantization and related entropy coding;
- Wavelet lossless compression for color and gray images;
- Progressive transmission algorithm for color bit compression;
- Progressive transmission and decompression algorithms;
- Non-uniform image compression algorithm;
- Quality based wavelet coefficient quantization tables;
- Attached optional post-processing filters;
- Image map editor;
- Optional peak signal noise ratio controlled compression;
- Special split and merge wavelet compression algorithm for very big image compression without any boundary effects ;
- Image dependent parameter optimization .

2. Main steps of the algorithm

In Lsic 3.0, three kinds of different image compression methods are included:

Method 1: Quality controlled wavelet based compression

Method 2: Color bit depth compression

Method 3: Wavelet lossless compression

Section 2.1-2.3 will give brief description of above method. The details will discuss later.

2.1. Main step of method 1

Figure 1 and Figure 2 give the flow charts of the image compression and decompression of method1 respectively. Every step in both compression and decompression has lot of details, which will be described later.

(2)

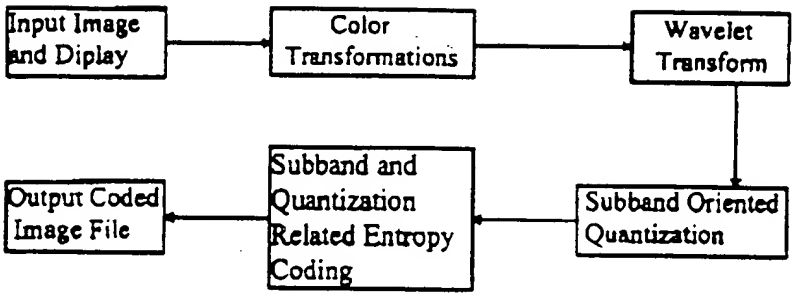


Figure 1: Compression flow chart for Method 1

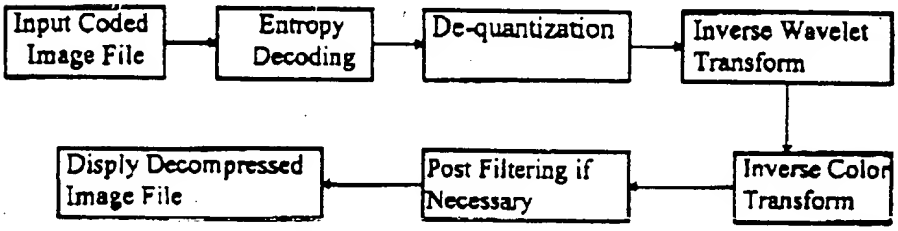


Figure 2: Decompression flow chart for Method 1

2.2. Main step of method 2

This method based on using less number of colors to approximately represent original images. Following figure gives the main step of the algorithm for the compression and decompression.

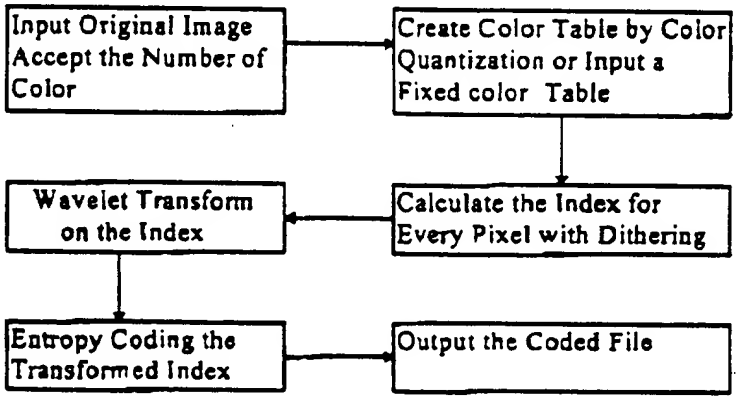


Figure 3: Compression flow chart for Method 2

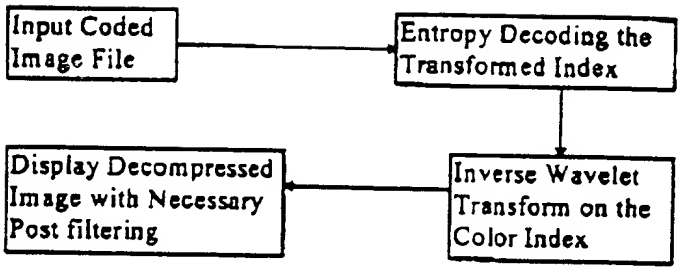


Figure 4: Decompression flow chart for Method 2

2.3. Main steps of method 3

The main steps of method 3 is almost as same as method 1. However, at the every step we use different methods. Following are its compression and decompression flow charts:

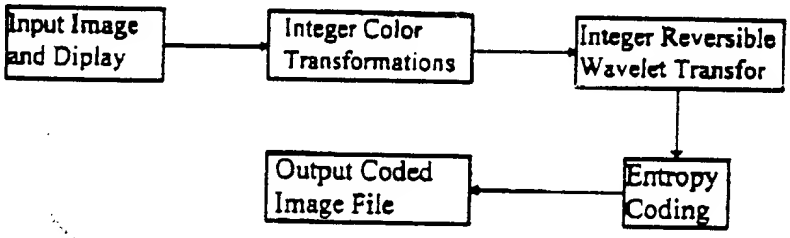


Figure 5: Compression flow chart for Method 3

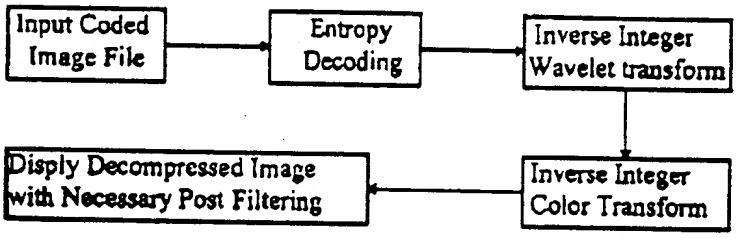


Figure 6: Decompression flow chart for Method 3

3. Brief Description for Other Features

Most Features (or inventions) are included in above three compression methods. Following is the brief introduction for some inventions list above.

(4)

3.1. Progressive decompression

This algorithm allows users to decode images from the lowest resolution to highest resolution. The advantage of this feature is that users can download small piece of the coded file and view the image at lower resolution to determine if they want to download the whole image.

Steps:

(a) Input the lowest pass component LL^0 of the coded file and reconstruct the lowest resolution image I^0;

(b) Display image I^0. If the user doesn't like it or the resolution is big enough for , stop; otherwise, go to next step;

(c) Input the lowest three band-pass components HL^0, LH^0 and HH^0 successively in the current coded file. Reconstruct the new image I^1 from LL^0, HL^0, LH^0 and HH^0. Let $I^0 = I^1$, go to step (b).

3.2. Non-uniform image compression

The algorithm allows users to divide the image into several parts with different interests and compress these areas with different qualities. The areas can have any shape. This algorithm is only available for method 1.

Original image goes though all of the procedure except quantization part, which follows the steps below:

(a) Creating the bitmap matrices related to the areas chosen by the user;

(b) Wavelet transform to every bitmap matrix;

(c) Different quantization in different areas according to the transformed matrices obtained above step.

3.3. Peak Signal Noise Ratio (PSNR) controlled compression

Peak Signal Noise Ratio (PSNR) is an image quality measurement used by most professional people. PSNR controlled compression allows users to choose their desired PSNR for the compressed image.

The related algorithm is an iterated system:

(a) Picking an initial parameter setting P_0;

(b) Quantize the wavelet coefficients with P_0 and calculate the corresponding PSNR;

(c) If the PSNR is close to desired one, stop and output the coded file; otherwise, get an adjusted vector ΔP_0 and set $P_0 \leftarrow P_0 + \Delta P_0$, go to step (b);

3.4. Attached optional post-processing filters

Users can choose any number of following processing filters at their compressing time. The desired results can stored in the coded image file, and, anyone who decompress the coded file will see the same result immediately.

- Sharpening images
- Smoothing images
- Improving the visual quality
- Brightening the images

3.5. Image map editor

Image Map Editor create a image map over Lsic3.0 compressed image file. User selects one or several areas of compressed image, assigns the http links to the areas, then, Image Map Editor calculates the coordinates of the areas and outputs a HTML associate with the image. User can add such information into his/her source code.

Following is an example of such image map:

<EMBED SRC="cow.cod" type = "image/cis-cod" WIDTH= "257" poly= "44, 45, 103, 78, 103, 86, 54, 86, 54, 78", href= "http://www.infinop.com"></EMBED>

3.6. Split and merge wavelet algorithm for very big image compression

This algorithm allows users to compress very big image by an ordinary machine. The key is to divide the original image into several smaller pieces and compress/decompress them separately by using overlap and dis-overlap technique. With this technique, the compression/decompression piece by piece is equal to compress/decompress the whole image together, which means users won't see any edge effect at decompressed image which appears at general split method.

Also, with this algorithm, users can either decompress the whole image or choose the specific part to decompress according to a image map we create for the division .

3.7. Image dependent optimized parameter setting

This algorithm allows user to get the best (or almost best) image quality at the desired compression ratio by choosing image related parameter setting.

3.8. Integer reversible wavelet algorithm with PPP property

See attached unpublished paper titled as "An Approach to Fast Integer Reversible Wavelet Transformations for Image Compression"

(6)

3.9. Integer color transformation

This algorithm is a integer reversible transform which has been used in lossless color image compression (Method 3) for Lsic 3.0.

The algorithm transform RGB color components to a new set of color components Y-Nb-Nr:

Forward transform RGB to Y-Nb-Nr:

$$Y = G + \mathrm{Int}\left(\frac{R+B}{2}\right),$$
$$Nb = B - \mathrm{Int}\left(\frac{Y}{2}\right),$$
$$Nr = R - \mathrm{Int}\left(\frac{Y}{2}\right).$$

Inverse transform Y-Nb-Nr to RGB:

$$R = Nr + \mathrm{Int}\left(\frac{Y}{2}\right),$$
$$B = Nb + \mathrm{Int}\left(\frac{Y}{2}\right),$$
$$G = Y - \mathrm{Int}\left(\frac{R+B}{2}\right).$$

3.10. Subband related Quantization and entropy coding

This entropy coding method is just designed for wavelet based image compression. The main idea is to take the advantage of different quantization at different subbands and encode each band according to its content. This method reduce the coding cost greatly.

CIS-1 Image Compression Algorithm

HONGYANG CHAO*
Computer and Information Science Inc.

1. Introduction

CIS-1, which has been being used in Lightning Strike image compression software, is a wavelet based image compression algorithm. CIS-1 has following advantages:

- o Reach almost optimal compression ratio;
- o Keep the major characteristics as more as possible. In other words, it reduce insignificant components gradually according to human visual system, so that people can still accept the image quality at the extremely high compression ratio;
- o Fast.

2. Main steps of the algorithm

Figure 1 and Figure 2 give the flow charts of the image compression and decompression respectively. Every step in both compression and decompression has lot of details, which will be described later.

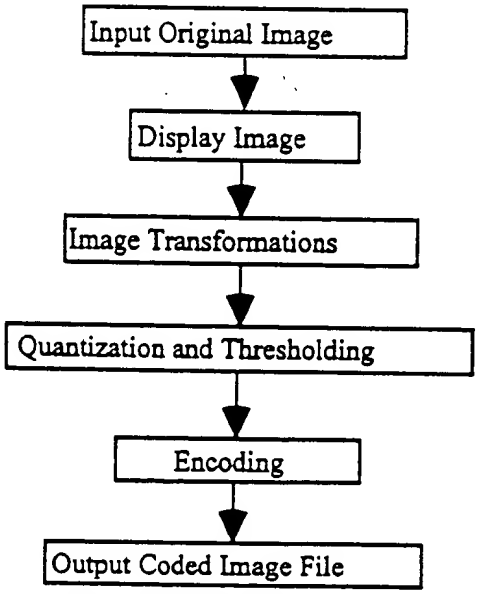


Figure 1: Image compression

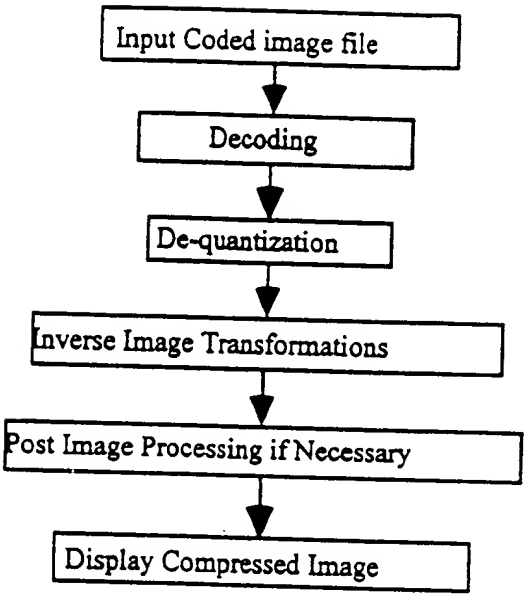


Figure 2: Image decompression

3. Image Compression

For the compression, we only have to describe three parts: image transformations, quantization and thresholding, and entropy coding.

3.1. Image Transformations

The image transformations involved in this algorithm include color transform (for color images) and wavelet decomposition (for both gray level images and color images).

(a) Color transform

In general, input color images are based on RGB color model, such as TIFF or BMP images. In order to get high compression ratio, it is better to change RGB color model to other color models, such as YIQ or YUV models.

RGB to YIQ:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

RGB to YUV:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.148 & -0.289 & 0.439 \\ 0.615 & -0.515 & -0.1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}.$$

(b) Wavelet decomposition

The purpose of wavelet transform is to represent the original image by different basis to achieve the objective of decorrelation. There are a lot of wavelets which can be used in this step. In CIS-1, we use a wavelet which results in the following algorithm: Suppose $\mathbf{C}^0 = \left[c_{jk}^0\right]_{M\times N}$ $(j = 0,\cdots,M-1;\ \ k = 0,\cdots,N-1)$ is original image, where M and N are integers which have the common factor 2^L (L is a positive integer). After one-level wavelet decomposition, we will get four parts as shown in figure 3.

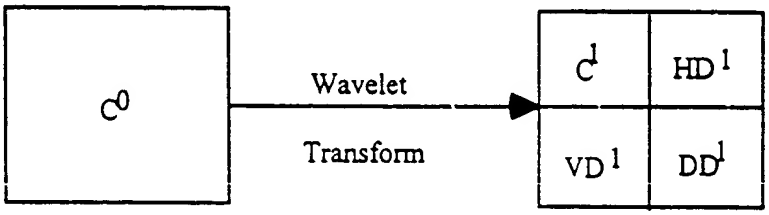


Figure 3. Wavelet image decomposition

We call $\mathbf{C}^1 = \left[c_{jk}^1\right]$ $(j = 0,\cdots,\frac{M}{2}-1;\ k = 0,\cdots,\frac{N}{2}-1)$ the blurred image of $\mathbf{C}^0$, $\mathbf{HD}^1$ the horizontal high frequency part of $\mathbf{C}^0$, $\mathbf{VD}^1$ the vertical high frequency part, and $\mathbf{DD}^1$ the diagonal ones. Setting $\mathbf{C}^0 = \mathbf{C}^1$, we can repeat the same procedure L times or until the size of the new blurred image $\mathbf{C}^1$ is small enough. Therefore, we only have to give the algorithm for one level decomposition:

(1) Let $\bar{\mathbf{C}}^0 = r\mathbf{C}^0$, $r>0$ is a factor which can be changed for different needs.

(2) Transform for image columns:

- For $k = 0,\cdots,N-1$, calculate

$$\begin{cases} \tilde{d}_{0k}^1 = \dfrac{\bar{c}_{1k}^0 - \bar{c}_{0k}^0}{2}, \\ \tilde{d}_{jk}^1 = \dfrac{1}{4}(\bar{c}_{2j-1,k}^0 - 2\bar{c}_{2j,k}^0 + \bar{c}_{2j+1,k}^0),\ \ j = 1,\cdots,\frac{M}{2}-1. \end{cases} \tag{3.1.1}$$

- For $k = 0,\cdots,N-1$, calculate

$$\begin{cases} \tilde{c}^1_{0,k} = \tilde{c}^0_{1,k} - \dfrac{\tilde{d}^1_{0,k} + \tilde{d}^1_{1,k}}{2}, \\ \tilde{c}^1_{j,k} = \tilde{c}^0_{2j+1,k} - \dfrac{\tilde{d}^1_{j,k} + \tilde{d}^1_{j+1,k}}{2}, \quad j = 1, \cdots, \frac{M}{2} - 2, \\ \tilde{c}^1_{\frac{N}{2}-1,k} = \tilde{c}^0_{N-1,k} - \tilde{d}^1_{\frac{N}{2}-1,k}. \end{cases} \tag{3.1.2}$$

(3) Transform for rows:

o For $j = 0, \cdots, \frac{M}{2} - 1$, computing

$$\begin{cases} hd^1_{j,0} = \dfrac{\tilde{c}^1_{j,1} - \tilde{c}^1_{j,0}}{2}, \\ hd^1_{j,k} = \dfrac{1}{4}(\tilde{c}^1_{j,2k-1} - 2\tilde{c}^1_{j,2k} + \tilde{c}^1_{j,2k+1}), \quad k = 1, \cdots \frac{M}{2} - 1. \end{cases} \tag{3.1.3}$$

and

$$\begin{cases} c^1_{j,0} = \tilde{c}^1_{j,1} - \dfrac{hd^1_{j,0} + hd^1_{j,1}}{2}, \\ c^1_{j,k} = \tilde{c}^1_{j,2k+1} - \dfrac{hd^1_{j,k} + hd^1_{j,k+1}}{2}, \quad k = 1, \cdots, \frac{M}{2} - 2, \\ c^1_{j,\frac{N}{2}-1} = \tilde{c}^1_{j,N-1} - hd^1_{j,\frac{N}{2}-1}. \end{cases} \tag{3.1.4}$$

o For $j = 0, \cdots, \frac{M}{2} - 1$, computing

$$\begin{cases} dd^1_{j,0} = \dfrac{\tilde{d}^1_{j,1} - \tilde{d}^1_{j,0}}{2}, \\ dd^1_{j,k} = \dfrac{1}{4}(\tilde{d}^1_{j,2k-1} - 2\tilde{d}^1_{j,2k} + \tilde{d}^1_{j,2k+1}), \quad k = 1, \cdots, \frac{N}{2} - 1. \end{cases} \tag{3.1.5}$$

and

$$\begin{cases} vd^1_{j,0} = \tilde{d}^1_{j,1} - \dfrac{dd^1_{j,0} + dd^1_{j,1}}{2}, \\ vd^1_{j,k} = \tilde{d}^1_{j,2k+1} - \dfrac{dd^1_{j,k} - dd^1_{j,k+1}}{2}, \quad k = 1, \cdots, \frac{M}{2} - 2, \\ vd^1_{j,\frac{N}{2}-1} = \tilde{d}^1_{N-1,k} - dd^1_{j,\frac{N}{2}-1}. \end{cases} \tag{3.1.6}$$

(4) $\mathbf{C}^1 = \left[c^1_{j,k}\right]$, $\mathbf{HD}^1 = \left[hd^1_{j,k}\right]$, $\mathbf{VD}^1 = \left[vd^1_{j,k}\right]$ and $\mathbf{DD}^1 = \left[dd^1_{j,k}\right]$, $j = 0, \cdots, \frac{M}{2} - 1$; $k = 0, \cdots, \frac{M}{2} - 1$.

Remark: If it is necessary, we also can use matrix multiply

Wavalet Coefficient Image of l levels = $\mathbf{W}_l \mathbf{C}^0 \mathbf{W}_l^T$.

Here, W_l is the transform matrix for l level wavelet decomposition.

3.2. Thresholding and Quantization

Both thresholding and Quantization allow us to reduce accuracy with which the wavelet coefficients are represented when converting the wavelet decomposition to an integer representation. This can be very important in image compression, as it tends to make many coefficients zeros--especially those for high spatial frequencies.

After L level, for example L=3, wavelet decomposition, we get the wavelet coefficients of the original image as plotted in Figure 4:

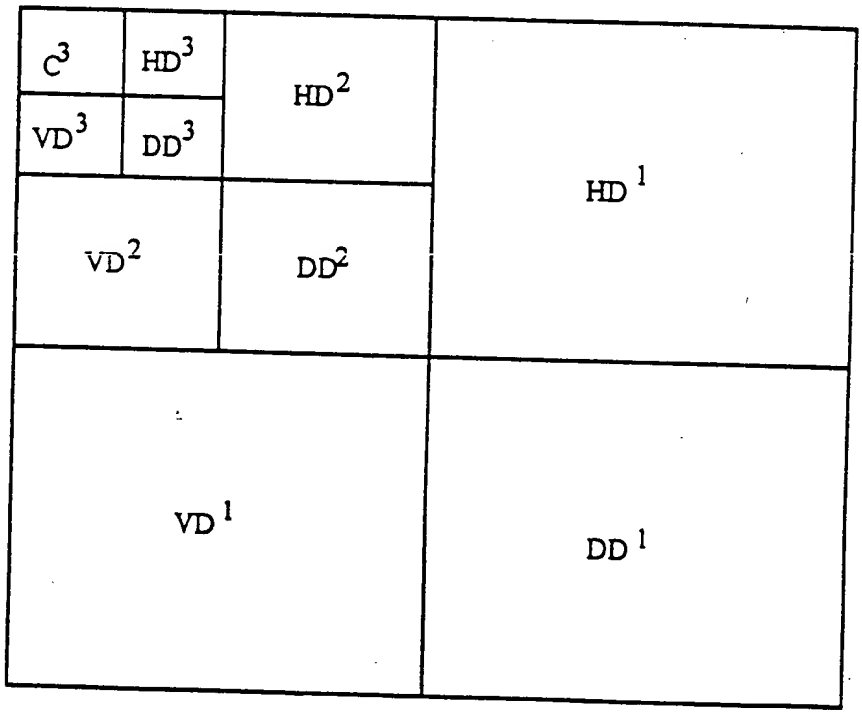


Figure 4. L=3 wavelet coefficients distribution

(a) Thresholding

In algorithm CIS-1, we use multilevel uniform thresholding method: Let

$$\mathbf{T} = (t_1, \cdots, t_L, t_{L+1})$$

be the chosen thresholds, where t_l is the threshold for l th $(l = 1, \cdots, L)$ level and t_{L+1} is a threshold for blurred image C^L. Thresholding is to set every entry in the blocks C^L,

$\mathbf{HD}^l$, $\mathbf{VD}^l$ and $\mathbf{DD}^l$ $(l=1,\cdots,L)$ to be zeros if its absolute value is not greater than the corresponding threshold.

Remark. For color image, we can have three threshold vectors which correspond three different color bands, such as Y, I and Q.

(b) Quantization

Quantization is to scale the wavelet coefficients and truncate them to integer values. In CIS-1, we use the quantization table shown in Table 1 to implement it.

q_{HD}^1	q_{HD}^2	$\bullet\bullet\bullet$	q_{HD}^L	
q_{VD}^1	q_{VD}^2	$\bullet\bullet\bullet$	q_{VD}^L	q_C^{L+1}
q_{DD}^1	q_{DD}^2	$\bullet\bullet\bullet$	q_{DD}^L	

Table 1. Quantization table

Here, the entries q_{HD}^l are quantization factors for blocks $\mathbf{HD}^l$ $(l=1,\cdots,L)$, q_{VD}^l and q_{DD}^l for blocks $\mathbf{VD}^l$ and $\mathbf{DD}^l$ $(l=1,\cdots,L)$ respectively, and the factor q_C^{L+1} is for the most blurred image $\mathbf{C}^L$. All the factors are integers between 0 and 255. The quantization scheme for the block $\mathbf{HD}^l$ $(l=1,\cdots,L)$ is

$$\overline{hd}_{j,k}^l = round\left(\frac{hd_{j,k}^l \bullet q_{HD}^l}{\max_{HD}^l}\right),\ j=0,\cdots,\tfrac{M}{2^l}-1;\quad k=0,\cdots,\tfrac{N}{2^l}-1. \quad (3.2.1)$$

Here, $\overline{hd}_{j,k}^l$ $(j=0,\cdots,\frac{M}{2^l}-1;\quad k=0,\cdots,\frac{N}{2^l}-1)$ are quantized wavelet coefficients in block $\mathbf{HD}^l$ $(l=1,\cdots,L)$,

$$\max_{HD}^l = \max_{\substack{0\le j\le (M/2^l-1)\\ 0\le k\le (N/2^l-1)}} \left(\left|hd_{j,k}^l\right|\right),$$

and the function *round* (x) gives the nearest integer of x. The scheme of quantization for other blocks are the similar to (3.2.1).

Remark. For color image, as the same as thresholding, we can have three separate quantization tables for different color bands.

3.3. Entropy Coding

Here, the encoding means the lossless compression for the wavelet coefficients. It is divided into two parts: Coefficient arrangement and entropy coding (Huffman or arithmetic).

4. Decompression

4.1 Decoding

Decoding, just as encoding, can be divided into two parts: Entropy decoding (Huffman or arithmetic), and coefficient rearranging.

4.2 Dequantization

After Decoding, we get quantized wavelet coefficients in $3*L+1$ Blocks. Dequantizing uses the same quantization table as quantizing, and the scheme as follow: for $l = 1, \cdots, L$

$$\widetilde{hd}^l_{j,k} = \frac{\widetilde{hd}^l_{j,k} \bullet \max^l_{HD}}{q^l_{HD}}, \; j = 0, \cdots, \frac{M}{2^l} - 1; \quad k = 0, \cdots, \frac{N}{2^l} - 1. \qquad (4.2.1)$$

(4.2.1) allows us to get the approximate coefficients for the blocks $\mathbf{HD}^l$ $(l = 1, \cdots, L)$, which is shown in Figure 4 . The dequantizing scheme for other blocks are similar to (4.1.2).

4.3 Inverse Image Transformations

(a) Wavelet reconstruction

We are going to describe the algorithm for one-level reconstruction which is plotted in Figure 5.

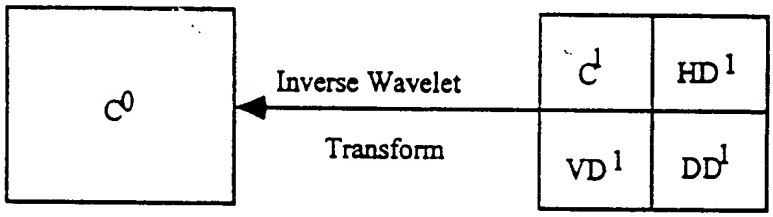


Figure 5. One-level wavelet reconstruction

(1) Inverse transform for rows:

- For $j = 0, \cdots, \frac{M}{2} - 1$, calculate

$$\begin{cases} \tilde{d}^1_{j,1} = vd^1_{j,0} + \dfrac{dd^1_{j,0} + dd^1_{j,1}}{2} \\ \tilde{d}^1_{j,2k+1} = vd^1_{j,k} + \dfrac{dd^1_{j,k} - dd^1_{j,k+1}}{2}, \quad k = 1, \cdots, \frac{N}{2} - 2, \\ \tilde{d}^1_{N-1,k} = vd^1_{j,\frac{N}{2}-1} + dd^1_{j,\frac{N}{2}-1}. \end{cases} \qquad (4.3.1)$$

and

$$\begin{cases} \tilde{d}^1_{j,0} = \tilde{d}^1_{j,1} - 2dd^1_{j,0}, \\ \tilde{d}^1_{j,2k} = \dfrac{(\tilde{d}^1_{j,2k-1} + \tilde{d}^1_{j,2k+1})}{2} - 2dd^1_{j,k}, \quad k = 1, \cdots, \frac{N}{2} - 1. \end{cases} \tag{4.3.2}$$

o For $j = 0, \cdots, \frac{M}{2} - 1$, calculate

$$\begin{cases} \tilde{c}^1_{j,1} = c^1_{j0} + \dfrac{hd^1_{j0} + hd^1_{j1}}{2}, \\ \tilde{c}^1_{j,2k+1} = c^1_{jk} + \dfrac{hd^1_{jk} + hd^1_{jk+1}}{2}, \quad k = 1, \cdots, \frac{N}{2} - 2, \\ \tilde{c}^1_{j,N-1} = c^1_{j,\frac{N}{2}-1} + hd^1_{j,\frac{N}{2}-1}. \end{cases} \tag{4.3.3}$$

and

$$\begin{cases} \tilde{c}^1_{j0} = \tilde{c}^1_{j1} - 2hd^1_{j,0}, \\ \tilde{c}^1_{j,2k} = \dfrac{1}{2}(\tilde{c}^1_{j,2k-1} + \tilde{c}^1_{j,2k+1}) - 2hd^1_{jk}, \quad k = 1, \cdots \frac{N}{2} - 1. \end{cases} \tag{4.3.4}$$

(2) Inverse transform for column:

o For $k = 0, \cdots, N-1$, calculate

$$\begin{cases} \tilde{c}^0_{1,k} = \tilde{c}^1_{0k} + \dfrac{\tilde{d}^1_{0k} + \tilde{d}^1_{1,k}}{2}, \\ \tilde{c}^0_{2j+1,k} = \tilde{c}^1_{jk} + \dfrac{\tilde{d}^1_{jk} + \tilde{d}^1_{j+1,k}}{2}, \quad j = 1, \cdots, \frac{M}{2} - 2, \\ \tilde{c}^0_{N-1,k} = \tilde{c}^1_{\frac{M}{2}-1,k} + \tilde{d}^1_{\frac{M}{2}-1,k}. \end{cases} \tag{4.3.5}$$

and

$$\begin{cases} \tilde{c}^0_{0k} = \tilde{c}^0_{1k} - 2\tilde{d}^1_{0k}, \\ \tilde{c}^0_{2j,k} = \dfrac{1}{2}(\tilde{c}^0_{2j-1,k} + \tilde{c}^0_{2j+1,k}) - 2\tilde{d}^1_{jk}, \quad j = 1, \cdots, \frac{M}{2} - 1 \end{cases} \tag{4.3.6}$$

(3) $c^0_{j,k} = \tilde{c}^0_{j,k} / r, \quad j = 0, \cdots, M-1; \quad k = 0, \cdots, N-1$. $\mathrm{C}^0 = \left[c^0_{j,k}\right]_{M \times N}$.

(b) Inverse color transform

For color image, we have to do inverse color transform

o YIQ to RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}.$$

$$\begin{cases} \tilde{d}^1_{j,0} = \tilde{d}^1_{j,1} - 2dd^1_{j,0}, \\ \tilde{d}^1_{j,2k} = \dfrac{(\tilde{d}^1_{j,2k-1} + \tilde{d}^1_{j,2k+1})}{2} - 2dd^1_{j,k}, \quad k = 1, \cdots, \frac{N}{2} - 1. \end{cases} \tag{4.3.2}$$

o For $j = 0, \cdots, \frac{M}{2} - 1$, calculate

$$\begin{cases} \tilde{c}^1_{j,1} = c^1_{j0} + \dfrac{hd^1_{j0} + hd^1_{j1}}{2}, \\ \tilde{c}^1_{j,2k+1} = c^1_{jk} + \dfrac{hd^1_{j,k} + hd^1_{j,k+1}}{2}, \quad k = 1, \cdots, \frac{N}{2} - 2, \\ \tilde{c}^1_{j,N-1} = c^1_{j,\frac{N}{2}} + hd^1_{j,\frac{N}{2}}. \end{cases} \tag{4.3.3}$$

and

$$\begin{cases} \tilde{c}^1_{j0} = \tilde{c}^1_{j1} - 2hd^1_{j,0}, \\ \tilde{c}^1_{j,2k} = \dfrac{1}{2}(\tilde{c}^1_{j,2k-1} + \tilde{c}^1_{j,2k+1}) - 2hd^1_{jk}, \quad k = 1, \cdots \frac{N}{2} - 1. \end{cases} \tag{4.3.4}$$

(2) Inverse transform for column:

o For $k = 0, \cdots, N - 1$, calculate

$$\begin{cases} \tilde{c}^0_{1,k} = \tilde{c}^1_{0k} + \dfrac{\tilde{d}^1_{0,k} + \tilde{d}^1_{1,k}}{2}, \\ \tilde{c}^0_{2j+1,k} = \tilde{c}^1_{jk} + \dfrac{\tilde{d}^1_{jk} + \tilde{d}^1_{j+1,k}}{2}, \quad j = 1, \cdots, \frac{M}{2} - 2, \\ \tilde{c}^0_{N-1,k} = \tilde{c}^1_{\frac{N}{2}-1,k} + \tilde{d}^1_{\frac{N}{2}-1,k}. \end{cases} \tag{4.3.5}$$

and

$$\begin{cases} \tilde{c}^0_{0,k} = \tilde{c}^0_{1k} - 2\tilde{d}^1_{0k}, \\ \tilde{c}^0_{2j,k} = \dfrac{1}{2}(\tilde{c}^0_{2j-1,k} + \tilde{c}^0_{2j+1,k}) - 2\tilde{d}^1_{jk}, \quad j = 1, \cdots, \frac{M}{2} - 1 \end{cases} \tag{4.3.6}$$

(3) $c^0_{j,k} = \tilde{c}^0_{j,k} / r, \quad j = 0, \cdots, M-1; \quad k = 0, \cdots, N-1$. $\mathbf{C}^0 = \left[c^0_{j,k}\right]_{\frac{M}{2} \times N}$.

(b) Inverse color transform

For color image, we have to do inverse color transform

o YIQ to RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.956 & 0.621 \\ 1.000 & -0.272 & -0.647 \\ 1.000 & -1.106 & 1.703 \end{bmatrix} \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}.$$

o YUV to RGB

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.140 \\ 1.000 & -0.395 & -0.581 \\ 1.000 & 2.032 & 0.000 \end{bmatrix} \begin{bmatrix} Y \\ U \\ V \end{bmatrix}$$

4.4 Necessary Post Image Processing

(a) Color Quatization

An Approach to Fast Integer Reversible Wavelet Transforms for Image Compression*

Hongyang Chao**
Computer and Information Science Inc.
3401 E. University, Suite 104, Denton, TX 76208
AND
Dept. of Computer Science, Zhongshan Univ.
Guangzhou 510275, P.R. China

Paul Fisher
Computer and Information Science Inc.
3401 E. University, Suite 104, Denton, TX 76208
AND
Dept. of Computer Science, Univ. of North Texas
Denton, TX 76205

Abstract

In this paper, we propose a general method for creating integer wavelet transforms which can be used in both lossless (reversible) and lossy compression of signals and images with arbitrary size. This method allows us to get a series of transformations which are very close to the corresponding biorthogonal wavelet transforms or some non-orthogonal wavelet transforms, but can be calculated with only integer addition and bit-shift operations. In addition, the integer wavelet transforms created in this paper possess a property of precision preservation (PPP). This property is very useful for conserving memory in both compression and decompression, and speed up the whole procedure in some applications. The motivation of this paper comes from the lifting scheme [1] and S+P transform [3].

* This work has been partially supported by the US Navy under SBIR Contract N00039-94-C-0013.

** The author is partially supported by the National Science Foundation of P. R. China and the Science Foundation of Zhongshan University.

1. Introduction

The wavelet transform has proven to be one of the most powerful tools in the field of image compression. Theoretically, the wavelet transformation is lossless, but since all computers have only finite precision, most of transformations are lossy in practice, even when we use floating point calculations. On the other hand, integer calculations are much faster than floating point for virtually all computers; and integer computations are much easier to implement in hardware which is more important in some applications. The memory utilization of integers is also a positive consideration. The difficulty is, if we directly use integers in the wavelet transform and its inverse without some proper considerations, it will cause the loss of accuracy. For some important image applications, the user wants to have complete control of the precision in which the image pixels are represented during the compression process, and thus prefers to have the image compressed from lossless to lossy.

Lossless compression is also very important for images found in such applications as medical and space science. In such situations, the designer of the compression algorithm must be very careful to avoid discarding any information that may be required or even useful at some later point. From the academic point of view, it is also very interesting to have a compression scheme which has very fast performance, and which can exactly reconstruct the image when necessary. In addition, it is also very useful to have some wavelet transforms which exhibit the property of precision preservation (PPP), which can be utilized in the computer which has limited precision and limited memory without losing any precision during the computation.

In this paper, we are going to describe two general methods from which one can get the integer wavelet transform desired. All of the wavelet transforms from the methods given in this paper possess the property of precision preservation (PPP). We draw on the work of several other authors who have already contributed to this area [2-3], where some specific examples were developed. However, this paper presents a more general method which allows one to see several new results as well as those presented and acknowledged prior to this work.

This paper is organized as follows: Section 2 and 3 give some examples of integer wavelet transforms. The examples in section 2 are the starting point for our approach, and the examples in Section 3 show the steps and motivation of our general method. Section 4 indicates how one can use the lifting technique to create an integer biorthogonal wavelet transform. The Correction technique to generate more general integer wavelet transforms is described in Section 5. Section 6 describes how to process boundaries in order to apply the integer calculation in finite sized images or signals. In Section 7, we prove the integer wavelet transforms developed by both the lifting and correction method possess the property of precision preservation (PPP). Some example images are also shown in this section. The last section, Section 8, provides the conclusion to this paper.

2. Basic integer wavelet transformations

We provide the following two examples as the starting point for our new method. For the sake of convenience, length, and simplicity, we only discuss the algorithm for a one level decomposition and reconstruction and only for a one dimensional signal. The extension to two dimensions is immediate as the rows and columns can be treated into a sequence of one dimensional signals. For the following examples, assume that $\{c_n^0\}_{n=0}^{N-1}$ is the original signal where the superscript indicates level and the subscript indicates a particular point in the signal. Also, $\{c_n^1\}_{n=0}^{N_1-1}$ and $\{d_n^1\}_{n=0}^{M_1-1}$ are its decomposition parts at the first level. Here

$$\begin{cases} N_1 = \begin{cases} \frac{N}{2}, & \text{if } N \text{ is an even number,} \\ \frac{N+1}{2}, & \text{if } N \text{ is an odd number;} \end{cases} \\ M_1 = N - N_1. \end{cases}$$

$\{c_n^1\}_{n=0}^{N_1-1}$ and $\{d_n^1\}_{n=0}^{M_1-1}$ are its low frequency (l) part and high frequency (h) part, respectively. For multi-levels, we just treat $\{c_n^1\}_{n=0}^{N_1-1}$ as $\{c_n^0\}_{n=0}^{N-1}$ and repeat the procedure again.

Example 1: A (2,2)-wavelet transform by integer calculation.

This transformation is similar to a variation of the Haar wavelet transform which uses low and high pass analysis (decomposition) filters given as:

n	0	1
$\tilde{h}_n$	1/2	1/2
$\tilde{g}_n$	1/2	-1/2

(1) Compute

$$d_k^1 = c_{2k}^0 - c_{2k+1}^0, \quad k = 0, \cdots, M_1 - 1. \tag{2.1}$$

(2) Compute

$$\begin{aligned} c_k^1 &= \mathrm{Int}\left(\frac{d_k^1}{2}\right) + c_{2k+1}^0, \quad k = 0, \cdots, N_1 - 2, \\ c_{N_1-1}^1 &= \begin{cases} \mathrm{Int}\left(\frac{d_{M_1-1}^1}{2}\right) + c_{N-1}^0, & \text{if } N \text{ is an even number,} \\ c_{N-1}^0, & \text{if } N \text{ is an odd number.} \end{cases} \end{aligned} \tag{2.2}$$

Here, $\mathrm{Int}(x)$ is an arbitrary rounding function which may have different interpretations. For example, $\mathrm{Int}(x)$ can be the integer which is nearest to x, or $\mathrm{Int}(x)$ may be any integer which satisfies $x - 1 < \mathrm{Int}(x) \le x$, etc. It is easy to see that all entries in both $\{c_n^1\}_{n=0}^{N_1-1}$ and $\{d_n^1\}_{n=0}^{M_1-1}$ are integers.

From (2.1)-(2.2), we can easily get the following integer reconstruction algorithm:

(b) Reconstruction

(1) If N is an even number, compute

$$c^0_{2k+1} = c^1_k - \mathrm{Int}(\frac{d^1_k}{2}), \quad k = 0, \cdots, N_1 - 1 \; ; \tag{2.3}$$

or, if N is an odd number, we have

$$\begin{aligned} c^0_{2k+1} &= c^1_k - \mathrm{Int}(\frac{d^1_k}{2}), \quad k = 0, \cdots, N_1 - 2, \\ c^0_{N-1} &= c^1_{N_1}. \end{aligned} \tag{2.4}$$

(2) Compute

$$c^0_{2k} = d^1_k + c^0_{2k+1}, \quad k = 0, \cdots, M_1 - 1. \tag{2.5}$$

Remark Since (2.1)-(2.6) are not linear because of the rounding operation $\mathrm{Int}(x)$, this means the transformation order becomes significant. For instance, if the decomposition was applied first to the columns and then to the rows, the inverse transformation must be applied first to the rows and then to the columns.

Example 2: Lazy wavelet transform.

The Lazy wavelet transform *does not do anything*. However, this illustrates an important concept. The corresponding inverse transform is nothing else but sub-sampling the even and odd indexed samples. Decomposition and reconstruction can use same formula as follows:

$$\begin{aligned} c^1_k &= c^0_{2k}, \; k = 0, \cdots, N_1 - 1; \\ d^1_k &= c^0_{2k+1}, \; k = 0, \cdots, M_1 - 1. \end{aligned}$$

Examples 1 and 2 are not good transforms for image compression, but they are simple. Much better transforms can be achieved from these two. As suggested above, we consider them only as a starting point for our integer, reversible, wavelet transform algorithm.

We must mention that there is another interesting property in the above two transforms which may not be easily seen. If the values of the signal pixels are represented by a finite number of bits, say one bit or one byte, we can still use the same number of bits to represent the result of the forward transform within the computer itself because of the complementary code property. While, from the reconstruction algorithm, the computer will get back the exact original signal through the same complementary code property. We call this property *a Property of Precision Preservation (PPP)* for these wavelets.

It is known that the general values of the high frequency wavelet coefficients are small, and all higher levels in the decomposition also provide generally small values in the high frequency band. This allows the preservation of precision during the computational stage of the wavelet coefficients. Now, the complementary code property, the other aspect of the *PPP* property is a well known characteristic of integer arithmetic as done by the computer. Consider the computation of the difference of two integers given as $c = b\text{-}a$ and the inverse computation of $a = b - c$. The nature of the computation within the computer can be specified as follows:

$$c_m = \begin{cases} b-a & \text{if } -2^{q-1} \le b-a < 2^{q-1}-1 \\ -2^q + b - a & \text{if } b-a \ge 2^{q-1} \\ 2^q + b - a & \text{if } b-a < -2^{q-1} \end{cases}$$

and the inverse is

$$a_m = \begin{cases} b-c_m & \text{if } -2^{q-1} \le b-c_m < 2^{q-1}-1 \\ -2^q + b - c_m & \text{if } b-c_m \ge 2^{q-1} \\ 2^q + b - c_m & \text{if } b-c_m < -2^{q-1} \end{cases}$$

where the m subscript indicates the internal representation, and the range of the integers a, b, c is $\left[-2^{q-1},\ 2^{q-1}-1\right]$. The internal representation of c_m when it is outside the range, its appearance is as a two's complement number, so the representation may not be the same as the external representation of c. However, the same complementary code for the a_m will cause the internal representation to be identical to the external representation of a. For example, if we let $b=2$ (00000010) and $a=-127$ (10000001) then c_m has the internal binary value of (10000001) when $q=4$. With a value of -127 for c_m, the inverse value for a_m will just be a.

In fact, for Example 2, this property is obviously true. While for Example 1, if the range of the pixel values is within a finite number of bits, say q, we can only use q bits as the working unit, which means the value of transform coefficients will also be within the interval with length 2^q, say $\left[-2^{q-1},\ 2^{q-1}-1\right]$. Due to the nature of computation on a machine, most machines will implement (2.1)-(2.2) automatically as follows (the complementary code property):

$$d_k^1 = \begin{cases} c_{2k}^0 - c_{2k+1}^0, & \text{if } -2^{q-1} \le c_{2k}^0 - c_{2k+1}^0 < 2^{q-1}, \\ c_{2k}^0 - c_{2k+1}^0 - 2^q, & \text{if } c_{2k}^0 - c_{2k+1}^0 \ge 2^{q-1}, \\ 2^q + \left(c_{2k}^0 - c_{2k+1}^0\right), & \text{if } c_{2k}^0 - c_{2k+1}^0 < -2^{q-1}. \end{cases} \tag{2.6}$$

$$c_k^1 = \begin{cases} \mathrm{Int}(\frac{d_k^1}{2}) + c_{2k+1}^0, & \text{if } -2^{q-1} \le \mathrm{Int}(\frac{d_k^1}{2}) + c_{2k+1}^0 < 2^{q-1}, \\ \mathrm{Int}(\frac{d_k^1}{2}) + c_{2k+1}^0 - 2^q, & \text{if } \mathrm{Int}(\frac{d_k^1}{2}) + c_{2k+1}^0 \ge 2^{q-1}, \\ \mathrm{Int}(\frac{d_k^1}{2}) + c_{2k+1}^0 + 2^q, & \text{if } \mathrm{Int}(\frac{d_k^1}{2}) + c_{2k+1}^0 < -2^{q-1}. \end{cases} \tag{2.7}$$

While the reconstruction algorithm (2.3) and (2.5) will be implemented by the computer itself as

$$c_{2k+1}^0 = \begin{cases} c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}), & \text{if } -2^{q-1} \le c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}) < 2^q, \\ 2^q + \left(c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}) \right), & \text{if } c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}) < -2^{q-1}, \\ \left(c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}) \right) - 2^q, & \text{if } c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}) > 2^{q-1}. \end{cases} \tag{2.8}$$

$$c_{2k}^0 = \begin{cases} d_k^1 + c_{2k+1}^0, & \text{if } -2^{q-1} \le d_k^1 + c_{2k-1}^0 < 2^{q-1}, \\ d_k^1 + c_{2k+1}^0 + 2^q, & \text{if } d_k^1 + c_{2k+1}^0 < -2^{q-1}, \\ d_k^1 + c_{2k+1}^0 - 2^q, & \text{if } d_k^1 + c_{2k+1}^0 \ge 2^{q-1}. \end{cases} \tag{2.9}$$

It is obvious that (2.8)-(2.9) are just the reverse of (2.6)-(2.7). It is also easy to see that if we properly take advantage of the bound in the coefficient size mentioned above, the algorithm can be implemented using a minimal amount of storage.

3. More Examples and Additional Analysis

In this section we are going to give more examples which will give some motivation for our new approach.

Example 3: A (2,6)-wavelet transform by integer calculation [2].

This transformation is similar to using following analysis filters

n	-2	-1	0	1	2	3
h_n	0	0	1/2	1/2	0	0
$\tilde{g}_n$	-1/16	-1/16	1/2	-1/2	1/16	1/16

(a) Decomposition

Decomposition starts with Example 1 at step (1) and (2), and then upgrades the high frequency component at step (3):

(1) Compute

$$d_k^{1,0} = c_{2k}^0 - c_{2k+1}^0, \quad k = 0, \cdots, M_1 - 1.$$

(2) Compute

$$c_k^1 = \mathrm{Int}(\frac{d_k^{1,0}}{2}) + c_{2k+1}^0, \quad k = 0, \cdots, N_1 - 2,$$

$$c_{N_1-1}^1 = \begin{cases} \mathrm{Int}(\frac{d_{M_1-1}^{1,0}}{2}) + c_{N-1}^0, & \text{if } N \text{ is an even number,} \\ c_{N-1}^0, & \text{if } N \text{ is an odd number.} \end{cases}$$

(3) Compute

$$\begin{cases} d_0^1 = \mathrm{Int}(\frac{c_0^1 - c_1^1}{4}) - d_0^{1,0} \\ d_k^1 = \mathrm{Int}(\frac{c_{k-1}^1 - c_{k+1}^1}{4}) - d_k^{1,0}, \quad k = 1, \cdots, M_1 - 2, \end{cases}$$

and then, if N is even, calculate

$$d_{M_1-1}^1 = \mathrm{Int}(\frac{c_{N_1-2}^1 - c_{N_1}^1}{4}) - d_{M_1-1}^{1,0},$$

else, calculate

$$d_{M_1-1}^1 = \mathrm{Int}(\frac{c_{N_1-3}^1 - c_{N_1}^1}{4}) - d_{M_1-1}^{1,0}.$$

(b) Reconstruction

The reconstruction algorithm is identical to the decomposition algorithm, except it is now running "backwards".

(1) Compute

$$\begin{cases} d_0^{1,0} = \mathrm{Int}(\frac{c_0^1 - c_1^1}{4}) - d_0^1 \\ d_k^{1,0} = \mathrm{Int}(\frac{c_{k-1}^1 - c_{k+1}^1}{4}) - d_k^1, \quad k = 1, \cdots, M_1 - 2, \end{cases}$$

and then, if N is even, calculate

$$d_{M_1-1}^{1,0} = \mathrm{Int}(\frac{c_{N_1-2}^1 - c_{N_1}^1}{4}) - d_{M_1-1}^{1,1},$$

else, calculate

$$d_{M_1-1}^{1,0} = \mathrm{Int}(\frac{c_{N_1-3}^1 - c_{N_1-1}^1}{4}) - d_{M_1-1}^{1,1}.$$

(2) If N is an even number, compute

$$c_{2k+1}^0 = c_k^1 - \mathrm{Int}(\frac{d_k^{1,0}}{2}), \quad k = 0, \cdots, N_1 - 1 ;$$

or, if N is an odd number, we have

$$c^0_{2k+1} = c^1_k - \mathrm{Int}(\frac{d^{1,0}_k}{2}), \quad k = 0, \cdots, N_1 - 2,$$
$$c^0_{N-1} = c^1_{N_1}.$$

(3) Compute

$$c^0_{2k} = d^{1,0}_k + c^0_{2k+1}, \quad k = 0, \cdots, M_1 - 1.$$

We see in step (2)-(3) above, that they are just the same as shown for the reconstruction of the (2,2)-wavelet transform (Example 1).

Example 4: A (1,3)-wavelet transform by integer calculation.
The following nonlinear transform is a variation of the transform which uses biorthogonal analysis filters:

n	-1	0	1
$\tilde{h}_n$	1	0	0
$\tilde{g}_n$	1/4	-1/2	1/4

(a) Decomposition

This decomposition starts with the *Lazy wavelet* at step (1) and upgrades the high frequency component at step (2):

(1) Set
$$c^1_k = c^0_{2k}, \ k = 0, \cdots, N_1 - 1;$$
$$d^1_k = c^0_{2k+1}, \ k = 0, \cdots, M_1 - 1.$$

(2) If N is an even number, calculate

$$\begin{cases} d^1_k = \mathrm{Int}(\frac{c^1_k + c^1_{k+1}}{2}) - d^{1,0}_k, \ k = 0, \cdots, M_1 - 2, \\ d^1_{M_1-1} = c^1_{N_1-1} - d^{1,0}_{M_1-1}. \end{cases}$$

Otherwise, if N is an odd number, calculate

$$d^1_k = \mathrm{Int}(\frac{c^0_{2k} + c^0_{2k+2}}{2}) - c^0_{2k+1}, \ k = 0, \cdots, M_1 - 1.$$

(b) Reconstruction

(1) Set
$$c^0_{2k} = c^1_k, \ k = 0, \cdots, N_1 - 1;$$

(2) If N is an even number, calculate

$$\begin{cases} c^0_{2k+1} = \mathrm{Int}(\frac{c^0_{2k} + c^0_{2k+2}}{2}) - d^1_k, \ k = 0, \cdots, M_1 - 2, \\ c^0_{N-1} = c^0_{N-2} - d^1_{M_1-1}. \end{cases}$$

Otherwise, if N is an odd number, calculate

$$c^0_{2k+1} = \text{Int}(\frac{c^0_{2k} + c^0_{2k+2}}{2}) - d^1_k, \quad k = 0, \cdots, M_1 - 1.$$

Example 5: A (5,3)-wavelet transform by integer calculation.

This transformation is also similar in function to using the biorthogonal analysis filters. It is given by

n	-2	-1	0	1	2
$\tilde{h}_n$	-1/8	1/4	3/4	1/4	-1/8
$\tilde{g}_n$	1/4	-1/2	1/4	0	0

(a) Decomposition

This decomposition starts with Example 3 at step (1) and upgrade low frequency components at step (2):

(1) Set $c^{1,0}_k = c^0_{2k}, \quad k=0, \cdots, N_1 - 1;$

If N is an even number, calculate

$$\begin{cases} d^1_k = \text{Int}(\frac{c^0_{2k} + c^0_{2k+2}}{2}) - c^0_{2k+1}, & k = 0, \cdots, M_1 - 2, \\ d^1_{M_1-1} = c^0_{N-2} - c^0_{N-1}. \end{cases}$$

Otherwise, if N is an odd number, calculate

$$d^1_k = \text{Int}(\frac{c^0_{2k} + c^0_{2k+2}}{2}) - c^0_{2k+1}, \quad k = 0, \cdots, M_1 - 1.$$

(2) If N is an even number, compute

$$\begin{cases} c^1_0 = c^{1,0}_0 - \text{Int}(\frac{d^1_0}{2}), \\ c^1_k = c^{1,0}_k - \text{Int}(\frac{d^1_{k-1} + d^1_k}{4}), & k = 1, \cdots, N_1 - 2, \\ c^1_{N_1-1} = c^{1,0}_{N_1-2} - \text{Int}(\frac{d^1_{N_1-2} + d^1_{N_1-1}}{4}). \end{cases}$$

Otherwise, if N is an odd number, calculate

$$\begin{cases} c_0^1 = c_0^{1,0} - \mathrm{Int}(\dfrac{d_0^1}{2}), \\ c_k^1 = c_k^{1,0} - \mathrm{Int}(\dfrac{d_{k-1}^1 + d_k^1}{4}), \quad k = 1, \cdots, N_1 - 2, \\ c_{N_1-1}^1 = c_{N_1-1}^{1,0} - \mathrm{Int}(\dfrac{d_{M_1-1}^1}{2}). \end{cases}$$

(b) Reconstruction

(1) Compute

$$c_0^0 = c_0^1 + \mathrm{Int}(\frac{d_0^1}{2}),$$

$$c_{2k}^0 = c_k^1 + \mathrm{Int}(\frac{d_{k-1}^1 + d_k^1}{4}), \quad k = 1, \cdots, N_1 - 2,$$

Then, if N is even, calculate

$$c_{N-2}^0 = c_{N_1-1}^1 + \mathrm{Int}(\frac{d_{N_1-2}^1 + d_{N-1}^1}{4}),$$

else calculate $$c_{N-1}^0 = c_{N_1-1}^1 + \mathrm{Int}(\frac{d_{M_1-1}^1}{2}).$$

(2) Compute $$c_{2k+1}^0 = \mathrm{Int}(\frac{c_{2k}^0 + c_{2k+2}^0}{2}) - d_k^1, \quad k = 0, \cdots, M_1 - 2,$$

Then, if N is even, calculate $$c_{N-1}^0 = c_{N-2}^0 - d_{M_1-1}^1.$$

The *PPP* property for Example 1-2 mentioned at the end of the previous section is also applicable for these three examples. It is obvious these three transformations are not really linear, but they are similar to the one using the corresponding filters given above. Especially, the filters in Example 3 and Example 5 belong to, with minor modification, the group of the best biorthogonal filters for image compression according to both our experience and the conclusion of [4].

Also, from the above three examples, we can note that if we begin with integer (linear or nonlinear) wavelet transformations and then use some proper upgrading formulas, we can get other, much better integer, wavelet transformations for image compression. Now, the key problem is: What kind of deductive formulas should be used? We provide an answer to this question in the following two sections, Section 4 and Section 5.

4. Lifting Scheme and Integer Biorthogonal Filtering

The *Lifting scheme*, discovered by Sweldens [1], is a new approach for constructing biorthogonal wavelets with compact support. However, the most interesting part of this method

for us is: it can be used, with minor modification, to create integer biorthogonal wavelet transformations. The following is an adaptation of the technique of [1].

Definition 1. The set of filters $\{h, \tilde{h}, g, \tilde{g}\}$ is a set of *biorthogonal filters* if the following formula is satisfied:

$$\forall \omega \in \mathbf{R}: \ \tilde{m}(\omega)\overline{m'(\omega)} = 1.$$

where
$$m(\omega) = \begin{bmatrix} h(\omega) & h(\omega+\pi) \\ g(\omega) & g(\omega+\pi) \end{bmatrix},$$

and
$$h(\omega) = \sum_k h_k e^{-ik\omega} \text{ and } g(\omega) = \sum_k g_k e^{-ik\omega},$$

and similarly for
$$\tilde{m}(\omega), \ \tilde{h}(\omega) \text{ and } \tilde{g}(\omega).$$

The following lemma is the main result of the *lifting scheme* [1] reported as corollary 6 in that paper.

Lemma 1. Take an initial set of finite biorthogonal filters $\{h, \tilde{h}^0, g^0, \tilde{g}\}$, then a new set of finite biorthogonal filters $\{h, \tilde{h}, g, \tilde{g}\}$ can be found as

$$\begin{aligned} \tilde{h}(\omega) &= \tilde{h}^0(\omega) + \tilde{g}(\omega)\overline{s(2\omega)} \\ g(\omega) &= g^0(\omega) - h(\omega)s(2\omega). \end{aligned} \tag{4.1a}$$

Similarly, if we take $\{h^0, \tilde{h}, g, \tilde{g}^0\}$ as an initial set of biorthogonal filters, a new set of finite biorthogonal filters $\{h, \tilde{h}, g, \tilde{g}\}$ can be found as

$$\begin{aligned} h(\omega) &= h^0(\omega) + g(\omega)\overline{\tilde{s}(2\omega)} \\ \tilde{g}(\omega) &= \tilde{g}^0(\omega) - \tilde{h}(\omega)\tilde{s}(2\omega). \end{aligned} \tag{4.1b}$$

Here, $s(\omega)$ is a trigonometric polynomial and the corresponding filter s is finite, and so is $\tilde{s}(\omega)$. Actually, regarding the filters, (4.1) is equivalent to

$$\begin{aligned} \tilde{h}_k &= \tilde{h}_k^0 + \sum_l \tilde{g}_{k+2l} s_l \\ g_k &= g_k^0 - \sum_l h_{k-2l} s_l. \end{aligned} \tag{4.2a}$$

or

$$\begin{aligned} h_k &= h_k^0 + \sum_l g_{k+2l} \tilde{s}_l \\ \tilde{g}_k &= \tilde{g}_k^0 - \sum_l \tilde{h}_{k-2l} \tilde{s}_l. \end{aligned} \tag{4.2b}$$

Next, we use the lifting scheme with minor modification to create an integer, nonlinear, quasi-biorthogonal, wavelet algorithm. Suppose $\{c_n^0\}$ is a original signal, $\{c_n^1\}$ and $\{d_n^1\}$ are again its low and high frequency decomposition parts, obtained by using the filters $\{h, \tilde{h}, g, \tilde{g}\}$.

If we use filters $\{\tilde{h}, \tilde{g}\}$ for decomposition (analysis), the corresponding decomposition algorithm is

$$\begin{cases} c_k^1 = \alpha_c \sum_n c_n^0 \tilde{h}_{n-2k}, \\ d_k^1 = \alpha_d \sum_n c_n^0 \tilde{g}_{n-2k}. \end{cases}$$

While the reconstruction algorithm will be

$$c_n^0 = 2\sum_k \left(\frac{c_k^1 h_{n-2k}}{\alpha_c} + \frac{d_k^1 g_{n-2k}}{\alpha_d} \right),$$

related to the synthesis filter $\{h, g\}$. Here, parameters α_c and α_d are positive constants with $\alpha_c \bullet \alpha_d = 2$. For example, in the situation of regular biorthogonal decomposition and reconstruction, $\alpha_c = \alpha_d = \sqrt{2}$; and for Example 1 through Example 5 above, $\alpha_c = 1$ and $\alpha_d = 2$.

If the set of filters $\{h, \tilde{h}, g, \tilde{g}\}$ is from $\{h, \tilde{h}^0, g^0, \tilde{g}\}$ by (4.2b), then the decomposition can be accomplished as follows:

1. Calculate

$$\begin{cases} c_k^{1,0} = \alpha_c \sum_n c_n^0 \tilde{h}_{n-2k}^0, \\ d_k^1 = \alpha_d \sum_n c_n^0 \tilde{g}_{n-2k}. \end{cases} \tag{4.3}$$

2. Calculate

$$c_k^1 = c_k^{1,0} + \frac{\alpha_c}{\alpha_d} \sum_l d_{k-l}^1 s_l. \tag{4.4}$$

The relative reconstruction scheme will be:

1. Calculate

$$c_k^{1,0} = c_k^1 - \frac{\alpha_c}{\alpha_d} \sum_l d_{k-l}^1 s_l, \tag{4.5}$$

2. Calculate

$$c_n^0 = 2\sum_k \left(\frac{c_k^{1,0} h_{n-2k}}{\alpha_c} + \frac{d_k^1 g_{n-2k}^0}{\alpha_d} \right). \tag{4.6}$$

Here, equations (4.3) and (4.6) are just the wavelet (inverse) transforms using biorthogonal filters $\{h, \tilde{h}^0, g^0, \tilde{g}\}$. While (4.4) and (4.5) are forward and backward upgrading formulas.

Similarly, if the set of filters $\{h, \tilde{h}, g, \tilde{g}\}$ is from the initial set of filters $\{h^0, \tilde{h}, g, \tilde{g}^0\}$ by using (4.2b), the relative decomposition is:

1. Calculate

$$\begin{cases} c_k^1 = \alpha_c \sum_n c_n^0 \tilde{h}_{n-2k}, \\ d_k^{1,0} = \alpha_d \sum_n c_n^0 \tilde{g}_{n-2k}^0. \end{cases}$$

2. Calculate $$d_k^1 = d_k^{1,0} - \frac{\alpha_d}{\alpha_c}\sum_l c_{k-l}^1 s_l .$$

The reconstruction scheme is:

1. Calculate $$d_k^{1,0} = d_k^1 + \frac{\alpha_d}{\alpha_c}\sum_l c_{k-l}^1 s_l$$

2. Calculate $$c_n^0 = 2\sum_k \left(\frac{c_k^1 h_{n-2k}^0}{\alpha_c} + \frac{d_k^1 g_{n-2k}}{\alpha_d} \right)$$

For the sake of clarity, we haven't considered the boundary situation, but we will address this later.

Corollary 4.1. Suppose biorthogonal filters $\{h, \bar{h}, g, \bar{g}\}$ are from initial filters $\{h, \bar{h}^0, g^0, \bar{g}\}$ by the lifting scheme (4.1a) or (4.2a). If the decomposition and reconstruction by filters $\{h, \bar{h}^0, g^0, \bar{g}\}$ can be accomplished only by integer calculation, such as Example 2, we also can create a corresponding integer wavelet decomposition and reconstruction scheme which is very "close" to the original one by using filters $\{h, \bar{h}, g, \bar{g}\}$. Here the word "close" means that the difference of the two decomposition schemes is just some rounding error, and this rounding error will be corrected by the integer reconstruction scheme.

In fact, if $\{c_k^{1,0}\}$ and $\{d_k^1\}$ are integer after (4.3), we can calculate $\{c_k^1\}$ by

$$c_k^1 = c_k^{1,0} + \mathrm{Int}\left(\frac{\alpha_c}{\alpha_d}\sum_l d_{k-l}^1 s_l \right). \tag{4.7}$$

instead of (4.4). Here $\mathrm{Int}(x)$, as described in Section 2, is an arbitrary rounding up function which satisfies $x-1 \le \mathrm{Int}(x) \le x+1$. It is obvious that (4.7) is very "close" to (4.4), and the exact reconstruction scheme can easily be obtained from

$$c_k^{1,0} = c_k^1 - \mathrm{Int}\left(\frac{\alpha_c}{\alpha_d}\sum_l d_{k-l}^1 s_l \right) \tag{4.8}$$

and (4.6). There will be a similar result, if the set of biorthogonal filters $\{h, \bar{h}, g, \bar{g}\}$ is obtained from the initial set of filters $\{h^0, \bar{h}, g, \bar{g}^0\}$ by using (4.2b).

We can now note, except for the example shown in the Lazy *wavelet*, (Example 2) most standard biorthogonal wavelet transforms cannot be performed directly by integer, even for one of the simplest wavelets, the *Haar wavelet*. However, if we properly choose the parameters α_c and α_d, and slightly change the transform algorithms, such as Example 1 and Example 3, we can have a variation of the original biorthogonal wavelet transforms with respect to the set of filters

$\{h, \tilde{h}^0, g^0, \tilde{g}\}$ (or $\{h^0, \tilde{h}, g, \tilde{g}^0\}$). On the other hand, the parameters $\{s_i\}$ should be also chosen carefully to guarantee that only addition and shift operations are needed by the algorithm.

Another observation: if the set of filters $\{h, \tilde{h}, g, \tilde{g}\}$ is obtained from a set of filters $\{h^0, \tilde{h}, g, \tilde{g}^0\}$ by the lifting scheme, and the set $\{h^0, \tilde{h}, g, \tilde{g}^0\}$ is also obtained from a filter set $\{h^0, \tilde{h}^0, g^0, \tilde{g}^0\}$, we can repeatedly use Corollary 1 to get a "close" integer wavelet transformation.

5. The Correction Method for Creating Integer Wavelet Transforms

In this section, we will describe another approach for obtaining integer wavelets by using the so called *Correction method*. The motivation of this method is from the S+P transform, and we will now generalize this approach. Actually, the lifting scheme for generating biorthogonal wavelets can be considered as a special case of the correction method. From this method we can get some even complicated filters with fast decomposition and reconstruction algorithm.

Suppose that we already have a simple integer wavelet transform, such as Examples 1 through 3, the decomposition and reconstruction scheme of which can be formulated as follows:

Decomposition

$$c_i^{1,0} = df_c\left(\{c_n^0\}\right)$$
$$d_i^{1,0} = df_d\left(\{c_n^0\}\right) \tag{5.1}$$

Reconstruction

$$c_n^0 = rf\left(\{c_i^{1,0}\}, \{d_i^{1,0}\}\right) \tag{5.2}$$

Here, (5.1) and (5.2) can be the same as (4.3) and (4.6) or other algorithms.

In general, after the above decomposition, one may not be satisfied with the result. There may still be some correlation among the highpass components because of the aliasing from the lowpass components, or the lowpass components do not carry enough of the expected information from the original signal. Hence, we could make an improvement by putting some correction part on the highpass components or lowpass components. There are many ways to accomplish this. However, for the sake of the integer calculation, we prefer to use following correction method. For example, if we want to make a correction for the highpass part, the corresponding formula would be:

$$d_i^1 = d_i^{1,0} - \mathrm{Int}\left(dc_i^1\right) \quad k = \cdots, 0, 1, 2, \cdots. \tag{5.3}$$

Here, dc_i^1 is a correction quantity for d_i^1

$$dc_i^1 = \sum_{m=s_0}^{s_1} \sigma_i c_{i+m}^{1,0} + \sum_{j=1}^{T} \tau_j d_{i+j}^{1,0}, \quad k = \cdots, 0, 1, 2, \cdots \tag{5.4}$$

and, $\{\sigma_i\}_{i=-S_0}^{S_1}$ and $\{\tau_j\}_{j=1}^{T}$ are given parameters which have been chosen for the user's purpose, such as reducing the redundancy among highpass components or some other special requirement. We are not going to discuss how to choose these parameters, but one can refer to the references [3, 5, 6] for clarification of this process. The only thing we need to mention is, for the sake of the integer calculation, any entries in both $\{\sigma_i\}_{i=-S_0}^{S_1}$ and $\{\tau_j\}_{j=1}^{T}$ should be rational numbers with denominators being powers of 2.

From (5.1), (5.3) and (5.4), it is easy to see the perfect reconstruction algorithm can be

$$d_k^{1,0} = d_k^1 + \mathrm{Int}(dc_k), \ k = \cdots, m, m-1, m-2\cdots, \tag{5.5}$$

combined with (5.2).

As mentioned above, the Lifting scheme is a special condition of the Correction method. Examples 3 through 5 can also be considered as the examples of this method. We next give an example of the Correction method which cannot be included in the group of Lifting scheme, and also which does not result in a closed form of compact support for biorthogonal filters.

Example 6 S+P transform [3], which is similar to using following analysis filters

n	-2	-1	0	1	2	3
$\tilde{h}_n$	0	0	1/2	1/2	0	0
$\tilde{g}_n$	-1/16	-1/16	15/32	-17/32	7/32	-1/32

While, the synthesis filters do not have compact support. However, the S+P transform can be implemented as follows:

(a) Decomposition

(1) Take the decomposition step of Example 1, that is, compute

$$d_k^{1,0} = c_{2k}^0 - c_{2k+1}^0, \quad k = 0, 1, \cdots, M_1 - 1;$$

and

$$c_k^1 = \mathrm{Int}\left(\frac{d_k^{1,0}}{2}\right) + c_{2k+1}^0, \quad k = 0, \cdots, N_1 - 2,$$

$$c_{N_1-1}^1 = \begin{cases} \mathrm{Int}\left(\frac{d_{N_1-1}^{1,0}}{2}\right) + c_{N-1}^0, & \text{if } N \text{ is an even number,} \\ c_{N-1}^0, & \text{if } N \text{ is an odd number.} \end{cases}$$

(2) Correction Step: Define $S_0 = -1$, $S_1 = 1$, $T = 1$ and

$$\sigma_{-1} = -\tfrac{1}{8}, \ \sigma_0 = -\tfrac{1}{8}, \ \sigma_1 = \tfrac{1}{8};$$
$$\tau_1 = \tfrac{1}{4}.$$

and now compute

$$\begin{cases} d_0^1 = d_0^{1,0} - \mathrm{Int}(\frac{c_0^1 - c_1^1}{4}); \\ d_k^1 = d_k^{1,0} - \mathrm{Int}(\frac{2c_{k-1}^1 + c_k^1 - 3c_{k+1}^1 - 2d_{k+1}^{1,0}}{8}), \quad k = 1, \cdots, M_1 - 2; \\ d_{M_1-1}^1 = d_{M_1-1}^{1,0} - \mathrm{Int}(\frac{c_{M_1-2}^1 - c_{M_1-1}^1}{4}). \end{cases}$$

(b) Reconstruction

(1) Compute

$$\begin{cases} d_{M_1-1}^{1,0} = d_{M_1-1}^1 + \mathrm{Int}(\frac{c_{M_1-2}^1 - c_{M_1-1}^1}{4}); \\ d_k^{1,0} = d_k^1 + \mathrm{Int}(\frac{2c_{k-1}^1 + c_k^1 - c_{k+1}^1 - 2d_{k+1}^{1,0}}{8}), k = M_1 - 2, \cdots, 1; \\ d_0^{1,0} = d_0^1 + \mathrm{Int}(\frac{c_0^1 - c_1^1}{4}). \end{cases}$$

(2) If N is an even number, compute

$$c_{2k+1}^0 = c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}), \quad k = 0, \cdots, N_1 - 1$$

or, if N is an odd number, we have

$$c_{2k+1}^0 = c_k^1 - \mathrm{Int}(\frac{d_k^1}{2}), \quad k = 0, \cdots, N_1 - 2,$$
$$c_{N-1}^0 = c_{N_1}^1.$$

(3) Compute

$$c_{2k}^0 = d_k^1 + c_{2k+1}^0, \quad k = 0, \cdots, M_1 - 1.$$

6. Boundary Conditions

In the previous two sections, we did not show how to get the integer, wavelet transform at the boundaries of signals. However, for all of the examples given above, the boundaries have been considered. There are two issues dealing with boundary filtering if we use the *Lifting scheme* or the *Correction method* to generate the integer wavelet transformations. The first is how to process the boundaries which occur in the start-up wavelet transformations. The second is how to deal with the boundaries in the deductive formula. If the boundaries in the start-up wavelet transform have already been established, then those in the upgrading formula are easy to establish. In fact, for the *Lifting scheme*, the boundaries in both steps should be processed in the same way. While, for the Correction *method*, it is easy to see from (5.3)-(5.4) that one has more choices to process boundaries in the second step. Therefore, the only thing we need to discuss here is the process by which the boundaries in the start up wavelet transformations are established. Assume we begin with compact supported biorthogonal wavelets.

Suppose the original signal is $\{c_k^0\}_{-0}^{N}$. For creating integer biorthogonal wavelet transformations we can use the following symmetric extension [7]:

(1). If current biorthogonal filters have even length, we extend the boundaries of the signal as $c_{-k}^0 = c_{k-1}^0$, $k = 1,2,\cdots$;

(2). If the filters have odd length, we do the extension as $c_{-k}^0 = c_k^0$, $k = 1,2,\cdots$.

Example 1 through 5 use the boundaries give above. In Example 6, the start up wavelet transform uses the above boundaries, but in the upgrading step, another boundary filtering is used. In addition, for arbitrarily sized images or signals, one can use the same technique which we described in the above examples to deal with this condition.

7. Some Applications

Before talking about any applications of the integer wavelet transform given above, we first prove that a nice *property of precision preservation (PPP)*, which is similar to the one mentioned in Section 2, holds for both the Lifting and Correction upgrading technique. This property is very important for many applications.

Lemma 7.1 Suppose that our integer wavelet transform starts with a pair of biorthogonal filters with the *PPP* property discussed in Section 2, that is, (4.3) and (4.6) possess this property. Then, the same property will be preserved in the whole algorithm if we adopt the Lifting scheme to be the upgrading formula.

In other words, Lemma 7.1 states if we only use the working units with the same precision as the original signal or image to calculate the wavelet transform developed in Section 4, the equations (4.8) and (4.6) are still the backward operations of the equations (4.3) and (4.7).

Proof. Assume that we only use q bits to represent images or signals, say, the range of the pixel values is within $[-2^{r-1}, 2^{r-1}-1]$. According to the hypothesis of the lemma, the equations (4.3) and its inverse (4.6) have the *PPP* property. Therefore, what we have to verify here is that the equation (4.7) and its inverse (4.8) can preserve the same property. We rewrite (4.7) and (4.8) as follow:

$$c_k^1 = c_k^{1,0} + b_k, \tag{7.1}$$

and its inverse

$$c_k^{1,0} = c_k^1 - b_k. \tag{7.2}$$

Here,

$$b_k = \mathrm{Int}\left(\frac{\alpha_c}{\alpha_d}\sum_l d_{k-l}^1 s_l\right).$$

In fact, the quantity b_k would have the same value in both (4.7) and (4.8) if we calculate it in the same way. On the other hand, if the working unit for b_k is q bits, the machine will give b_k another value, say $\tilde{b}_k$ ($-2^{q-1} \le \tilde{b}_k < 2^{q-1}$), where $\tilde{b}_k$ is not equal to b_k in the sense of mathematics if the value of b_k is beyond the interval $[-2^{q-1}, 2^{q-1}-1]$. However, $\tilde{b}_k$ will be the same in both (4.7) and (4.8). Therefore, the machine will automatically implement (7.1) and (7.2) as

$$c_k^1 = \begin{cases} c_k^{1,0} + \tilde{b}_k, & \text{if } -2^{q-1} \le c_k^{1,0} + \tilde{b}_k < 2^{q-1}, \\ c_k^{1,0} + \tilde{b}_k - 2^q, & \text{if } c_k^{1,0} + \tilde{b}_k \ge 2^{q-1}, \\ 2^q + c_k^{1,0} + \tilde{b}_k & \text{if } c_k^{1,0} + \tilde{b}_k < -2^{q-1}. \end{cases} \tag{7.1m}$$

and

$$c_k^{1,0} = \begin{cases} c_k^1 - \tilde{b}_k, & \text{if } -2^{q-1} \le c_k^1 - \tilde{b}_k < 2^{q-1}, \\ 2^q + c_k^1 - \tilde{b}_k, & \text{if } c_k^1 - \tilde{b}_k < -2^{q-1}, \\ c_k^1 - \tilde{b}_k - 2^q, & \text{if } c_k^1 - \tilde{b}_k \ge 2^{q-1}. \end{cases} \tag{7.2m}$$

It is easy to see that (7.2m) is just the backward operation of (7.1m), which provides the evidence that the conclusion of this lemma is correct.

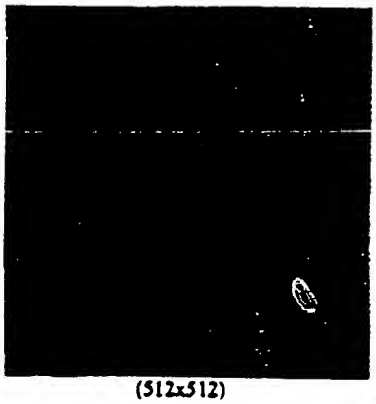
It should be mentioned that the coefficients $\{c_k^1\}$ obtained by (4.3) and (7.1m) might not be the "real" wavelet coefficients using common sense. However, if we still use the working unit with q bits precision at the reconstruction step, (7.2m) and (4.6) will give the exact original signal back. On the other hand, the coefficients $\{c_k^1\}$ still keep the most continuity of the "real" wavelet coefficients. Therefore, when we repeat the decomposition step on $\{c_k^1\}$, most small coefficients in its high frequency part $\{d_k^2\}$ will be almost the same as the "real" coefficients (within some rounding error), which allows us to still take advantage of the "real" wavelet transform in image compression.

A similar argument can show the same *PPP* property will hold for the integer wavelet transforms generated by the Correction method in Section 5.

As we mentioned before, for many applications, the lossless image compression is as important as lossy compression. The integer wavelet transforms give the opportunity to compress without loss. It is also obvious that the integer wavelet algorithms can be used wherever ordinary wavelets are used, especially in signal and image compression. However, for most computers, the integer wavelet transform is much faster than the ordinary one and it uses much less memory. The following are some applications illustrating these types of transforms.

Application 1. Lossless image compression

As mentioned at the beginning of this paper, the integer wavelet transformation established by the techniques described in this paper can always be used for lossless image compression because of the reversible ability. Especially, we can use the *PPP* property discussed in Lemma 7.1. We have used this wavelet lossless technology (WLT) for gray scale lossless image compression, and we have tried several images. For most natural images, the size of wavelet lossless compressed images is much smaller than corresponding GIF images. Figure 1 through 4 give some examples. Figure 1 is a standard image for compression, Figure 2 and 4 are X-ray images and Figure 3 is a two-value image but we treat it as a 8 bit gray scale image in order to compare with the GIF format. In fact, if we convert Figure 2 to a binary image, better result can be obtained by the JBIG technique.



(512x512)
Figure 1. Compression Ratio
WLT: 1.9:1/GIF: 1.05:1

(512x512)
Figure 2. Compression Ratio:
WLT 4.5:1/GIF: 2.72:1

"Visioneer may have come up with on
against the paper blizzard. . .gets pile:
way to others throughout your compar

Figure 3. Compression Ratio: WLT: 20.8/ GIF 17.8 (152x794)

9-41

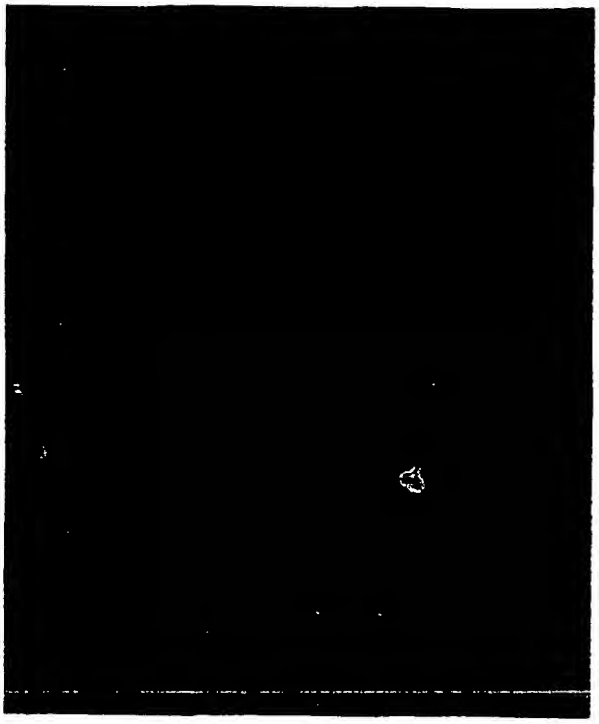


Figure 4. Compression Ratio: WLT 3.8:1/GIF 1.98:1 (1232x1024)

Application 2. Large scale medical image compression

Usually, 12 bits are used to represent one pixel in medical images. In this situation, the values of the pixels vary from 0 to 4095. Such images require careful treatment when a transform coding method is used for compression. If we use ordinary biorthogonal wavelets, the range of the transform coefficients will expand to $[-2^{16}, 2^{16}]$ when five levels of transform are used. Therefore, a longer working unit has to be employed, which consumes significant computer resources. However, the integer wavelet technique developed in this paper will solve this problem. For example, if we use the transforms given in Example 3, 5 and 6, the values of transform coefficients will be limited to the range of $[-2^{13}, 2^{13}]$. Even if we do not use the *PPP* property for these wavelets, *16* bits for the working unit is sufficient for all computations.

8. Conclusion

This paper has shown the processes necessary in order to obtain a non-linear, integer, biorthogonal, or non-biorthogonal reversible wavelet transform suitable for signal or image processing. We have shown how such a transform can be obtained either using the Lifting method, or the Correction method. For example, all interpolation wavelets can be modified to be corresponding integer wavelets without loosing any properties of original wavelets. In addition,

3-42

we have shown under certain conditions, the precision of the transform computation on the computer can remain at the same precision of the data, thus reducing the need for additional computer memory during the transform computation. These are extremely powerful techniques when the target data are large images, or the requirements establish a need for speed.

Although this paper establishes the structure for the integer transform based upon the biorthogonal wavelet or some non-biorthogonal wavelet, we do not imply the examples in this paper are necessarily the best wavelets for any particular application. However, we do claim if one is going to use such a technique, the ideas suggested in this paper will provide the best implementation.

References

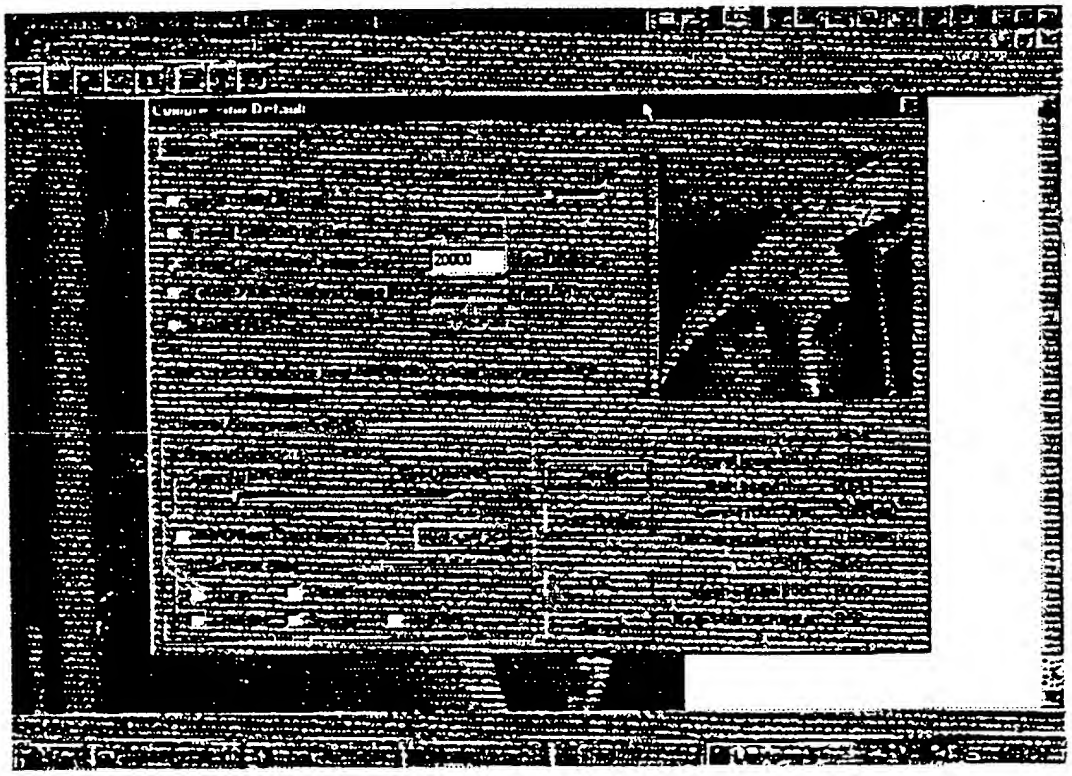
1. Wim Sweldens, *The lifting scheme: A custom-design construction of biorthogonal wavelets*, Applied and Computational Harmonic Analysis, Vol. 3, No. 2, April 1996.
2. A. Zandi, J. Allen, E. Schwartz and M. Boliek, *CREW: Compression with reversible embedded wavelets*, in IEEE Data Compression Conference, (Snowbird, Utah), pp.212-221, March 1995.
3. Amir Said, *An image multiresolution representation for lossless and lossy compression*, Submitted to the IEEE Transactions on Image Processing.
4. J. Villasenor, B. Belzer, and J. Liao, *Wavelet filter evaluation for image compression*, IEEE Trans. Image Processing, Vol. 4, pp.1053-1060.
5. G.R. Kuduvalli and R.M. Rangayyan, *Performance analysis of reversible image compression techniques for high-resolution digital teleradiology*, IEEE Trans. Med. Imaging, Vol. 11, pp. 430-445, Sept. 1992.
6. W.H. Press, B.P Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes: the Art of Science Programming*, Cambridge University Press, Cambridge, New York, 1986.
7. G. Strang and Truong Nguyen, *Wavelets and filter banks*, Wellesley-Cambridge Press, 1996.

INFINITRON

Windows Image Compressor V3.0

Lightning Strike

Announcing, Lightning Strike™ Image Compressor (LSIC) version 3.0, a Windows 95 tool that compresses still images from 50:1 to 200:1 using INFINITRON's proprietary wavelet technology. LSIC is a versatile, easy to use tool for Web and Graphic designers that can handle a wide variety of digital image formats, and it includes filters and convenient web tools. Images can be compressed 3 to 5 times more than JPEG, while maintaining similar or better image fidelity. Images can be viewed in 2 to 4 seconds over the Internet rather than 10 to 20 seconds for images compressed under JPEG. This has enormous benefits for reducing bottlenecks on corporate networks and the web, and in addition, requires less storage space.



Lightning Strike Features

Compression. Images can be compressed as high as 200:1 using wavelet technology.

Compression Control. An EASY mode allows the user to compress images with minimal input, requiring only a decision between more quality or more compression. An ADVANCED mode enables the user to select; 1) image file size, 2) compression ratio, 3) PSNR, or 4) master level. Web designers will like the one step process to control the size of their image files, thus insuring the speed an image may be viewed on a browser.

Non Uniform Compression. Regions of an image can be selected for less compression to preserve a higher image quality while the rest of the image is compressed to the specified compression ratio. In this way important parts of a picture maintain crucial details while the overall picture file can be made as small as possible.

Post Reconstruction Filters. Filters are available to enhance the reconstructed image. At compression time the user can preset a control to have these filters operate automatically during reconstruction. The filters include; quality improvement, sharpen (edge enhancement), smoothing, and brighten.

Transparencies. The user will have the ability to set pixels transparent so that a color in the background (already on the page) can be seen through the picture. This is useful for creative web site developers.

Progressive Compression. An image can be compressed so that when it is viewed it will appear quickly, first with low resolution, and then progressively building up in detail as it is downloaded. This insures the viewer does not lose interest while the image is downloaded.

INFINITRON

Windows Image Compressor V3.0

Lightning Strike

Cow, Bitmap, No Compression

Cow, Lightning Strike, 85:1 compression

Cow, JPEG 85:1 Compression

Applications Lightning Strike Image Compression

- Images on the Web
- Photo Stock
- Data Warehousing
- Catalogues
- Video Games
- CDs (Encyclopedias, Museums, Science, and Medicine)
- Archives (Art, History, Genealogy)
- Medical Imaging

Performance

Encode time typically less than 3 seconds for a 320 X 240 pixel, 24 bit color image on a 133 MHz. Pentium with 16 MB RAM.

Minimum Recommended System

Windows 95/NT OS
Pentium 100 MHz, 8 MB RAM
2 MB for program files
10 MB plus to swap image files

Ancillary INFINITRON Products

- Netscape Navigator Plug-in
- Java Applet
- ActiveX Control
- Web Site Image Converter
- Lightning Strike SDK
- GML Banner Generator

About INFINITRON, Inc.

Founded in 1992, INFINITRON, Inc. is a private company that specializes in the design and marketing of high quality image and video compression solutions for a wide array of markets. INFINITRON is based in Vancouver, BC with labs in Regina, Saskatchewan and Denton Texas.

Download a FREE demo version of Lightning Strike Windows Compressor from:
www.infinitron.com

INFINITRON USA Office 3401 East University, #104, Denton, TX, 76208
Tel: 817.484.1165 FAX: 817.484.0588

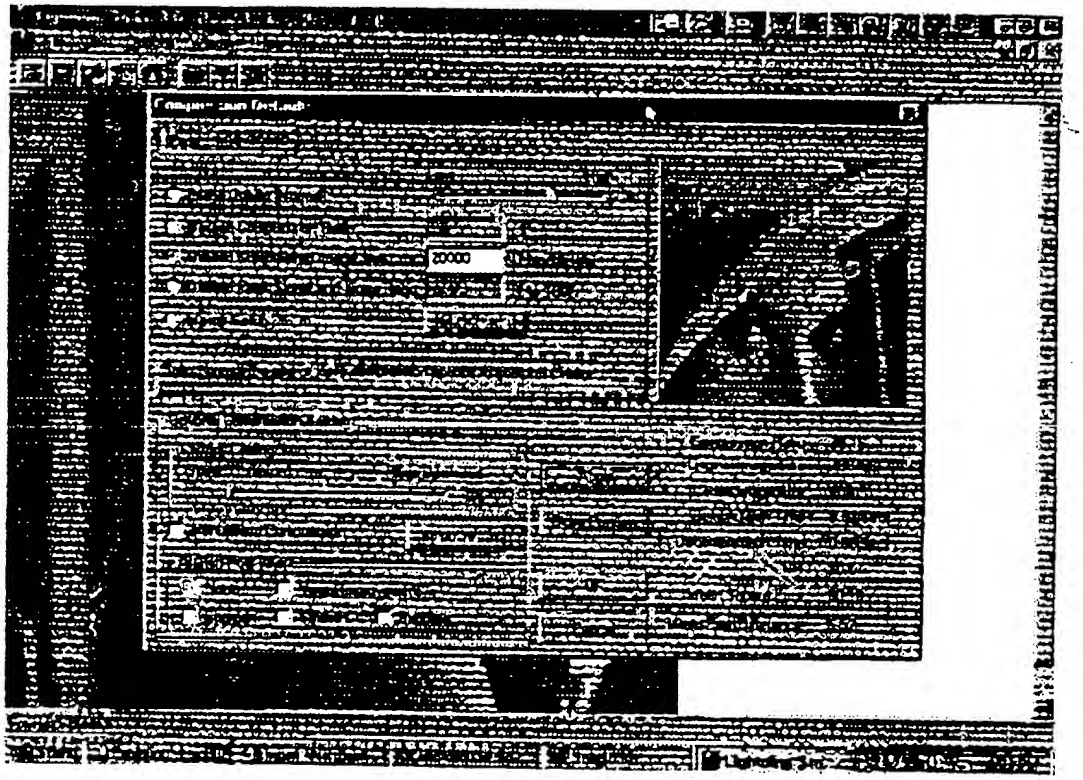
INFINITRON Canada Office 10th Fl. 1199 W. Hastings, Vancouver, BC V6E 3T5
Tel: 604.688.9789 FAX: 604.688.9798

INFINITRON

Windows Image Compressor V3.0

Lightning Strike

Announcing, Lightning Strike Image Compressor (TM) (LSIC) version 3.0, a Windows 95 tool that compresses still images from 50:1 to 200:1 using Infinitron's proprietary Wavelet technology. LSIC is a versatile, easy to use tool for Web and Graphic designers that can handle a wide variety of digital image formats and includes filters, and convenient web tools. Images can be compressed to files 4 times smaller than JPEG, while maintaining similar or better image fidelity. Images can be viewed in 2 to 5 seconds over the internet rather than 10 to 20 seconds for images compressed under JPEG. This has enormous benefits for reducing bottlenecks on corporate networks and the web, and in addition, requires less storage space.



Lightning Strike Features

Compression. Images can be compressed to ~~over 100~~:1 using Wavelet Technology. 200:1

Compression Control. An EASY mode allows the user to compress images with minimal input, requiring only a decision between more quality or more compression. An ADVANCED mode enables the user to select; 1) image file size, 2) compression ratio, 3) PSNR, or 4) master level. Web designers will like the one step process to control the size of their image files, thus insuring the speed an image may be viewed on a browser.

Non Uniform Compression. Regions of an image can be selected for less compression to preserve a higher image quality while the rest of the image is compressed to the specified compression ratio. In this way important parts of a picture maintain crucial details while the overall picture file can be made as small as possible

Post Reconstruction Filters. Filters are available to enhance the reconstructed image. At compression time the user can preset a control to have these filters operate automatically during reconstruction. The filters include; quality improvement, sharpen (edge enhancement), smoothing, and brighten.

Transparencies. The user will have the ability to set pixels transparent so that a color in the background (already on the page) can be seen through the picture. This is useful for creative web site developers.

Progressive Compression. An image can be compressed so that when it is viewed it will appear quickly, first with low resolution, and then progressively building up in detail as it is downloaded. This insures the viewer does not loose interest while the image is downloaded.

INFINITRON

Windows Image Compressor V3.0

Lightning Strike

Lenna .bmp, No Compression

Lenna, JPEG 115:1 Compression

Applications Lightning Strike Image Compression

- Images on the Web
- Photo Stock,
- Data Warehousing
- Catalogues.
- Video Games
- CDs (Encyclopedias, Museums, Science, and Medicine)
- Archives (Art, History, Geneology)
- Medical Imaging

Performance

Encode time typically less than 3 seconds for a 320 X 240 pixel, 24 bit color image on a 133 MHz. Pentium with 16 MB RAM.

Minimum Recommended System

Windows 95/NT OS
Pentium 100 MHz, 8 MB RAM
2 MB for program files
10 MB plus to swap image files

Ancillary INFINITRON Products

- Netscape Navigator Plug-in
- Java Applet
- ActiveX Control
- Web Site Image Converter
- Lightning Strike SDK
- GMT. Banner Generator

About INFINITRON, Inc.

Founded in 1992, INFINITRON, Inc. is a private company that specializes in the design and marketing of high quality image and video compression solutions for a wide array of markets. INFINITRON is based in Vancouver, BC with labs in Regina, Saskatchewan and Denton Texas.

Download a FREE demo version of Lightning Strike Windows Compressor from:
www.infinitron.com

INFINITRON USA Office: 3401 East University, #104, Denton, TX, 76208
Tel: 817.484.1165 FAX: 817.484.0588

INFINITRON Canada Office: 10th Floor 1199 W. Hastings, Vancouver, BC, V6E 3T5
Tel: 604.688.9789 FAX: 604.688.9798

INFINITRON
Lightning Strike

Product Fact Sheet
Windows Compressor

Compressor Version 3.0

The Lightning Strike Compressor is a Windows tool that compresses still images from a wide variety of digital image formats using Infinitron's proprietary Wavelet algorithm. Images can be compressed to files 5 times smaller than JPEG, while maintaining similar or better image fidelity. Images can be viewed in 1 or 2 seconds over the internet rather than 10 to 20 seconds for images compressed under JPEG. This has enormous benefits for transmitting over corporate networks or the web, and in addition, saves space required for storing all those images.

Lightning Strike is a collection of tools in a user friendly environment. Two levels of user control are offered, one quick and easy for most applications, the other a master level for the advanced user who wishes to control parameters to maximize image quality.

The compression approach used by Lightning Strike is based upon integer wavelets. This technology is acknowledged by leading experts as a superior compression technique as compared to discrete cosine transform used in JPEG.

Lightning Strike Windows Compressor Features

Image Compression Options and Control

Compression Technique Options

Both Infinitron's Wavelet Compression and other frequently used compression methods are included in the product so users need only have Lightning Strike on their work station to perform all image compressions. Images can be compressed to Wavelet, JPEG, PNG, and GIF.

Compression Quality Versus Speed Options

The user can select one of two encoding processes that trade quality for speed of compression and ease of use. With the "Advanced" option selected, the optimum compression parameters are set by the user to give the best possible images for selected compression ratio. With "Easy" selected you get the fastest compression without having to know details of parameter selection.

Compression Ratio Control

The compressed image file size or compression ratio may be specified rather than the quality factor. This enables a web designer to control the size of their image files or the speed an image may be viewed, in a one step process.

Region of Interest Focusing

Regions of an image can be selected for less compression to preserve a higher image quality while the rest of the image is compressed to the specified compression ratio. In this way important parts of a picture maintain crucial details while the over all picture file can be made as small as possible. This is also known as Non-Uniform Compression.

INFINITRON
Lightning Strike

Product Fact Sheet
Windows Compressor

Split and Merge

Very large images, which could not otherwise be compressed due to their large size, can be split into smaller images and compressed individually. This process has the side advantage of using RAM more effectively speeding time for compression. The split images can be reassembled using the merge aspect of the feature.

Post Reconstruction Filters

Filters are available to enhance the reconstructed image. At compression time the user can preset a control to have these filters operate automatically during reconstruction. The filters include; quality improvement, sharpen (edge enhancement), smoothing, and brighten.

Transparencies

The user will have the ability to set pixels transparent so that a color in the background (already on the page) can be seen through the picture. This is useful for creative web site developers. This gives the ability to display pictures other than the rectangular shape allotted on the web page, i.e. circles, polygons etc.. Also, designers often use this feature for shadowing, letters and objects.

Progressive Decompression

An image can be compressed so that when it is viewed it will appear quickly, first with low resolution, and then progressively building up in detail as it is downloaded. This insures the viewer does not loose interest while the image is downloaded.

INFINITRON
Lightning Strike

Product Fact Sheet
Windows Compressor

Image Comparisons

Picture of Lena, with no Compression (512 X 512 Image)

INFINITRON
Lightning Strike

Product Fact Sheet
Windows Compressor

Picture of Lena Compressed 100:1 with Lightning Strike

INFINITRON
Lightning Strike

Product Fact Sheet
Windows Compressor

File Functions

PNG File Structure
The compressed images are stored in file format compliant with the PNG standard. In the future this file format will replace the GIF format used today.

Batch Compress
The user is able to compress many images at once by adding or deleting image files (or paths) to a list box.

Image Statistics
The compressor stores image statistics on each compressed image which may be viewed by the user. The following information is provided: image dimensions, compression ratio, file sizes, MSE, PSNR, maximum pixel difference, compression and decompression times.

Performance and System Requirements

Encoding and Decode Time.
The typical time to encode or decode a 320X240, 24 bit color image is 1 second on a Pentium running at 133 MHz with 16 Meg RAM.

Minimum Recommended System
The minimum system requirements for an IBM PC Compatible are:

Hard Disc Drive	2 Mbytes free for program files, 10 Mb plus to swap image files.
Operating System	MS Windows 3.1(Win32)/ 95/ NT
RAM	8 Mbytes

This software is also available on the Apple MAC, Solaris, and UNIX platforms.

INFINITRON
Lightning Strike

Product Fact Sheet
Windows Compressor

Ancillary Infinitron Products

Netscape Navigator Plug-in
Netscape plug-ins are available for the Mac68k, Mac PPC, Widows 3.1 and 95/NT.

Java Applet
Java Applets are available for the Mac68k, Mac PPC, Widows 3.1 and 95/NT.

ActiveX Control
The Lightning Strike decompression software is available for such applications as Microsoft's Internet Explorer, as an ActiveX control.

Web Site Image Converter
This utility will automate the conversion of web pages from JPEG to the Lightning Strike format. The utility searches an HTML file and replicates it replacing any JPEG image tags with Lightning Strike tags and converting the JPEG image files to Lightning Strike. The utility can follow link tags to recursively convert and replicate an entire web site or sub-section of a web site to the Lightning Strike format. This utility will be available for Windows NT and most flavors of the UNIX operating system.

Lightning Strike Software Developers Kit
Using the SDK, a developer can integrate the highly efficient Lightning Strike module libraries into their own applications.

Download a FREE demo version of Lightning Strike Windows Compressor from: www.infinitron.com

Infinitron USA Office: 3401 East University, #104, Denton, TX, 76208
Tel: 817.484.1165 FAX: 817.484.0588

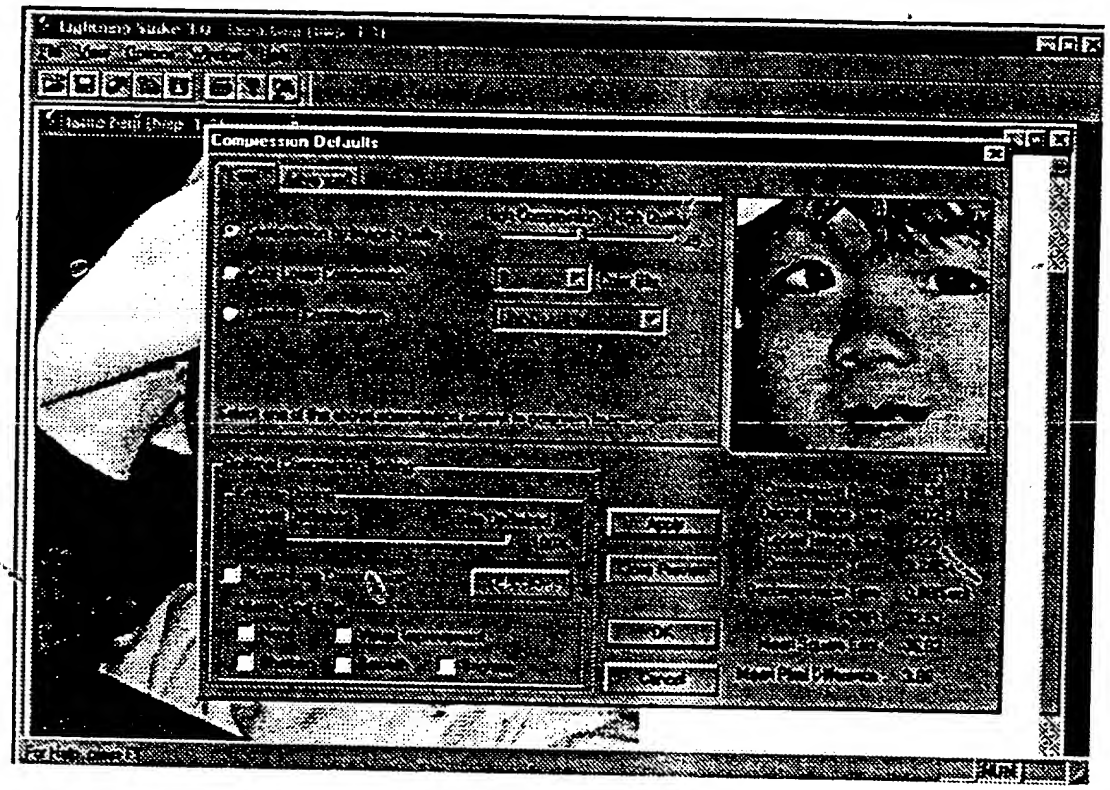
Infinitron Canada Office: 10th Flr 1199 W. Hastings, Vancouver, BC, V6E 3T5
Tel: 604.688.9789 FAX: 604.688.9789

INFINITRON

Windows Image Compressor V3.0

Announcing, Lightning Strike™ Image Compressor (LSIC) version 3.0, a Windows 95 tool that compresses still images from 20:1 to 200:1 using INFINITRON's proprietary wavelet technology. LSIC is a versatile, easy to use tool for Web and Graphic designers that can handle a wide variety of digital image formats, and it includes filters and convenient web tools. Images can be compressed at ratios well in excess of present JPEG ratios while maintaining comparable image fidelity. This translates to much shorter image down load time on the web. This has enormous benefits for reducing bottlenecks on corporate networks and the web, and in addition, requires less storage space.

Lightning Strike



Lightning Strike Features

Compression. Uses a proprietary integer wavelet.

Compression Control. An EASY mode allows the user to compress images with minimal input, requiring only a decision between quality and compression. An ADVANCED mode enables the user to select; 1) image file size, 2) compression ratio, 3) PSNR, or 4) master level for professionals where every parameter can be altered. We also provide the highest, wavelet lossless compression for users wishing this capability. Web designers will like the one step process to control the size of their image files allowing control over the delivery time of an image over a network.

Non Uniform Compression. Regions can be selected for less compression to preserve image quality while the rest of the image is compressed to the specified compression ratio. In this way, important parts of a picture maintain crucial details while the over all picture file can be made as small as possible

Post Reconstruction Filters. Filters are available to enhance the reconstructed image. At compression time the user can preset a control to have these filters operate automatically during reconstruction. The filters include; visual quality improvement, sharpen (edge enhancement), smoothing, and brighten.

Transparencies. The user will have the ability to set pixels transparent, so that a color in the background (already on the page) can be seen through the picture. This is useful for creative web site developers.

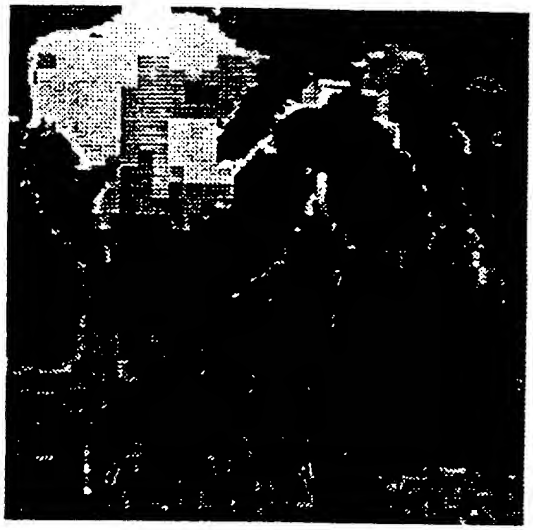
Progressive Compression. An image can be compressed so that when it is viewed it will appear quickly, first with low resolution, and then progressively building up in detail as it is downloaded. This insures the viewer does not lose interest while the image is downloaded.

INFINITRON

Windows Image Compressor V3.0

Lightning Strike

Cow, Bitmap, No Compression



Cow, JPEG 85:1 Compression

Cow, Lightning Strike, 85:1 compression

Applications for Lightning Strike Image Compression

- Images on the Web
- Photo Stock
- Data Warehousing
- Catalogues
- Video Games
- CDs (Encyclopedias, Museums, Science, and Medicine)
- Archives (Art, History, Genealogy)
- Medical Imaging

Performance

Encode time typically less than 2.5 seconds for a 640 X 480 pixel, 24 bit color image on a 133 MHz. Pentium with 16 MB RAM. Decode time is less than .75 seconds.

Minimum Recommended System

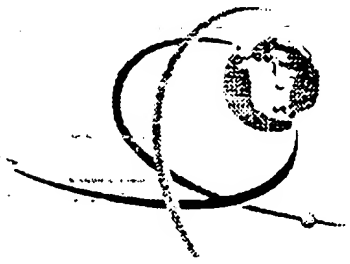
Windows 95/NT OS
Pentium 100 MHz, 8 MB RAM
2 MB for program files
10 MB plus to swap image files

Ancillary INFINITRON Products

- Netscape Navigator Plug-in
- Java Applet
- ActiveX Control
- Web Site Image Converter
- Lightning Strike SDK
- GML Banner Animation/Compression
- Black and White Image Compression

About INFINITRON, Inc.

Founded in 1992, INFINITRON, Inc. is a private company that specializes in the design and marketing of high quality image and video compression solutions for a wide array of markets. INFINITRON is based in Vancouver, BC with offices in Regina, Saskatchewan and Denton Texas.



Download a FREE demo version of Lightning Strike Windows Compressor from:
www.infinitron.com

INFINITRON USA Office — 3401 East University, #104, Denton, TX, 76208
Tel: 817.484.1165 FAX: 817.484.0586

INFINITRON Canada Office — 10th Flr 1199.W. Hastings, Vancouver, BC, V6E 3T5
Tel: 604.688.9789 FAX: 604.688.9798